TAMPERE UNIVERSITY OF TECHNOLOGY

Miro Nieminen

Fallback Mechanisms for Connection Loss in Single-Page Web Applications

Master of Science Thesis

Examiners: Tommi Mikkonen
Examiners and topic approved in the Information Technology Department Council meeting on 08.04.2015

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
Miro Nieminen: Fallback Mechanisms for Connection Loss in Single-Page Web Applications
Diplomityö, 53 sivua
Toukokuu 2015
Pääaine: Ohjelmistotuotanto
Tarkastajat: Tommi Mikkonen
Avainsanat: JavaScript, Web Application, Single-Page Application, Computer Supported Collaborative Work, Offline Support

Web-teknologioiden nopea kehittyminen niin työpöytä- kuin mobiililaitteissa on tehnyt selaimesta varteenotettavan sovellusalustan lähes kaikenlaisille ohjelmistoille. Suorituskykyisen selaimen löytyminen yhä useammasta taskusta tekee web-teknologioiden käyttämisestä yhä houkuttelevampaa ja kustannustehokkaampaa myös toteutettaessa liiketoimintakriittisiä sovelluksia.

Mobiililaitteiden määrän kasvuvauhti on ohittanut matkapuhelinverkkojen datayhteyksien kantokyvyn kasvuvauhdin. Laitteita myös käytetään yhä syrjäisemmissä sijainneissa, missä datayhteydet ovat rajallisia tai jopa olemattomia. Tämä aiheuttaa ongelmia käytettäessä web-sovelluksia, joiden toiminta on riippuvainen yhteydestä palvelimeen. Huono datayhteys tuottaa ongelmia web-sovelluksen käyttäjälle, kun sovellus saattaa olla hetkittäin täysin toimimattomassa tilassa.

Tässä työssä keskitytään siihen, miten web-kehittäjät voivat varautua yhteyden katkoksiin ja heikkoon laatuun sovellustasolla. Työn pohjana käytetään tapaustutkimusta, missä päiväkotiympäristössä käytettävään Päikky-sovellukseen lisätään offline-tuki. Tapaustutkimuksessa toteutettuja ratkaisuja arvioidaan käyttäjähaastattelujen avulla sen kannalta, miten ne sopivat yleispäteviksi ratkaisumalleiksi web-sovelluksissa yhteyskatkosten varalle. Tehtyjä ratkaisuja arvioidaan myös käytettävyysnäkökulmasta, ja siitä, miten ne tukevat keskivertokäyttäjää ja ovat tälle ymmärrettävissä.

Työn tulosten pohjalta esitetään suunnitteluperiaatteita sovelluskehittäjille yhteyskatkoksiin varautumista varten. Käyttäjähaastatteluissa ilmenneet epäkohdat ratkaisun käyttökokemuksesa listataan ja niihin ehdotetaan mahdollisia parannusehdotuksia.

Työn keskeisimmät tulokset osoittavat, että yhteysongelmiin tulee ainakin jollakin tasolla varautua nykypäivänä kaikissa liiketoimintakriittisissä web-sovelluksissa. Vaikka varsinaista offline-tukea ei toteutettaisikaan, voidaan työssä esitetyillä suunnitteluperiaatteilla parantaa huomattavasti web-sovelluksen käyttökokemusta.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
Miro Nieminen: Fallback Mechanisms for Connection Loss in Single-Page Web Applications
Master of Science Thesis, 53 pages
May 2015
Major: Software Engineering
Examiner: Tommi Mikkonen
Keywords: JavaScript, Web Application, Single-Page Application, Computer Supported Collaborative Work, Offline Support

Fast-paced evolution of web technologies in both desktop and mobile devices has made browser environment a reckoned platform for almost any kind of application. The fact that more and more people are carrying efficient browsers in their pockets makes usage of web technologies more tempting and cost efficient solution even when creating business critical applications.

The growth rate of mobile device usage has surpassed the rate in which new mobile network is built. The devices are also used in more distant locations where the data connection of the mobile network can be very limited or almost non-existent. This causes problems when using web applications, which are dependent on the connection to the server. Bad connectivity results in a degenerated user experience, since the application might be completely unusable when the connection is dropped.

In this thesis we focus on how web developers could prepare the application for connection loss on the application level. The research is based on a case study, in which Päikky, an application from the kindergarten domain, is implemented with an offline support. The solutions done on the case study's offline support implementation are evaluated with user interviews from a technical viewpoint and from the user experience perspective. The emphasis on the evaluation is that could the solutions be generalized as a design guidelines for offline support, and are the solutions understandable and usable for an average user.

Based on the results of the research a set of design guidelines for offline support implementation are defined. The user experience flaws found in the user interviews are listed, and possible solutions for them are discussed.

The essential results of this thesis indicate that connection issues are something that application should be prepared for, at least if the application is business critical. Even if there is no need for a full offline support, following the guidelines introduced will improve any web application's user experience significantly.

PREFACE

Academic writing surely is not for everyone, I can tell you that. The booklet you are holding (or watching via screen) is the result of the mentally hardest project I have ever pulled through. Even if I never actually doubt myself about finishing this thesis, there were some dark moments during the creation of this ensemble.

This space is usually used for thanking relevant people. And by coincidence also I have several people to thank for making this thesis happen.

I would like to thank my examiner Professor Tommi Mikkonen, who has the amazing ability to make the writing of a thesis to sound always so easy and straightforward task. The various hands-on advices which made the writing easier were also highly appreciated.

The completion of this thesis is a fact thanks to also my instructor D.Sc. Sami Vihavainen, to whom I surely was not the most optimal thesis worker to mentor. Even after three weeks of full-time thesis work which resulted only under 5 pages of text Sami could find the positive sides of the work done and encourage me to go further.

Massive thanks goes also to my employee Futurice, which supplied the extraordinarily awesome circumstances for the creation of this thesis. Especially I HAVE to thank the allmighty tribe Tammerforce, since without the peer pressure provided by them this thesis would still be in the making.

Lastly I would like to send love to my home team J and D, who cheered me through this project and withstanded my constant tantrums during it.

And now, as they say in Finland: *"Torille!"*

Tampere, April 17, 2015 Miro Nieminen

CONTENTS

1. INTRODUCTION

The world is digitalizing around us in an increasing a pace. More and more of our work tasks and everyday activities are affiliated to digital devices and services. Most of these things are often useless without connectivity to a broader context, usually the Internet.

Age of the desktop hegemony is already over. Globally over a third of the Internet usage is done with mobile devices (smartphones and tablets) [1]. Unlike as in the desktop devices, which usually are operated from static location aided by physical access to the Internet, mobile devices often travel with their owners where ever they might go. This creates challenges for connectivity which is nowadays essential for the proper usage of the device.

The problem with mobile devices is that mobile broadband coverage varies within different locations. When a mobile device is not in the coverage area of a Wireless Local Area Network, the device relies on a possible mobile broadband connection. When moving away from the nearest base station the reception of the signal weakens. Mobile device user experiences this as an increased latency times, lower data transferring speeds, or even as a total blackout of the broadband service. When moving away from highly populated areas this is inevitable, since not all locations are to be covered properly while keeping cost efficiency on a sane level, especially in countries with large acreage but low population, like Finland. However digitalization of services is even more important on places like this, since supplying physical services to areas with a low population density is expensive.

This makes digital services more and more common in business critical tasks for a emerging amount of applications. But how to use services and devices dependent on the Internet with a coincidental connection? With the increasing mobile device usage, adding offline support at least in some level will not be an uncommon task for any web developer.

The Internet is build on top of the client-server -paradigm [2]. This means that the server holds the master data, and clients has to do all the data fetching and modifying through the server. Without connectivity to the Internet, the client has to manage with the content it might already have loaded. This is the basic problem surrounding connectivity issues – without access to the Internet, there is not much what the client can do.

1.1 Context

This thesis studies one real-life case of implementing offline support to already existing system created for supporting kindergarten personnel and the parents of daycare children, the Päikky system. Päikky is used by various kindergartens in Finland and its key feature is to provide passage control mechanism for the kindergarten's children.

The Päikky system is owned and managed by *MukavaIT Oy* (later referred to as the *client organization* on this thesis), a Finnish start-up company. The client organization has bought consulting and development to Päikky from *Futurice Oy*, where the author of this thesis has also worked as a part of the Päikky development team.

On a high level Päikky consists of four modules: *Family UI, Kindergarten UI, Manager UI* and the *backend*. Under this thesis' analysis is especially the *Kindergarten UI*, which is a mobile-first designed web application. Kindergarten nurses use the Kindergarten UI on smartphones every time a child enters or leaves the kindergarten. As this is an additional task to their current workload, it has to be as effortless as possible, especially when it is performed tens of times per day.

The Kindergarten UI is a web application built in accordance with the *single-page application* paradigm. In contrary to traditional web pages, where the server outputs the requested content as structured and stylized HTML content, single-page apps use mainly *REST APIs*[1] for the exchange of the data. With REST API after web application initialization only the actual information can be transfered between the client and the browser, making the client in charge for the rendering format and style of the data. This results in a business logic being duplicated on the browser, making the client more "fat". Since the logic exists side by side on the client as well as on the server, the client can be prepared to operate even when the connectivity to the server is temporarily lost.

In this thesis the emphasis will be on how to deal with the connection issue inside the browser environment, ignoring the possible solutions that could be done on the device platform tier. Therefore results of this thesis will apply for browsers in desktop computers, tablet devices and mobile phones, while leaving the network level solutions out of the scope.

1.2 Research Objectives

This thesis researches an implementation of an offline mode to an existing single-page application, and studies the possible missteps that were made. Ultimately this thesis aims to define design guidelines for both user experience and software development

[1] Representational State Transfer API

-wise, so other developers would be able to achieve offline support implementation with less work and potentially with better results.

The objective of this research is to find useful and working fallback mechanisms for connectivity loss in single-page web applications, in both from the user experience perspective and from the developer's viewpoint. The state where the application loses the connectivity to the Internet is referred on this thesis as an **offline mode**. In this thesis an artifact is implemented and evaluated which allows the user to continue their tasks when the connection is lost and the application enters the offline mode.

From software development point of view the question is, how to deliver seamless user experience, even when the connectivity is bad or nonexistent? How much resources can this reserve on the development phase? Is it possible to abstract the connection quality completely away and handle the issues with connection internally on the application level, invisible to the user?

From the user experience perspective thesis concentrates on how to notify the user about the loss of Internet connection. Does the user need to be notified at all? If the user is notified, what is exactly the message that needs to be told?

This thesis answers to the following question:

> ''How to develop fallback mechanisms efficiently for Internet connection shortages on a single-page web application to support user experience"

Based on the research, implementation and evaluation done on this thesis, a recommendations for other developers are presented on the form of design guidelines.

1.3 Structure of this Thesis

Chapter 2 opens up the background in which from the themes of this thesis are originating from. The research gap in which this thesis resides is also introduced. *Chapter 3* describes the real-life case which is in a vital role on this thesis. Parts that are essential for understanding the concepts discussed on this thesis are opened thoroughly. *Chapter 4* introduces the methods used to conduct the research done on this thesis. On *Chapter 5* technical solutions implemented are summed up and described. On *Chapter 6* these solutions are evaluated, from both technical viewpoint and from the user experience perspective. Evaluation is done based on user interviews conducted during this thesis. *Chapter 7* presents the design guidelines formed based on the evaluation done on the previous chapters. Also aspects noted during the evaluation but which were too indeterminate to be design guidelines are discussed on the same chapter. Conclusions done based on the implementation and the evaluation of the offline support are presented on the *Chapter 8*.

2. BACKGROUND

In this chapter the three prime themes around this thesis are introduced. Basic knowledge of these themes are required in order to fully comprehend all the aspects on the case study investigated on this thesis later on.

The basics of *web as an application platform* and more precisely the *single-page application* paradigm are introduced as it is the method how the web application is built on the case study. The concept of *computer supported collaborative work* is introduced since the case study's application is essentially that: multiple users collaboratively viewing and editing the same data set. The most usual types of *connectivity issues* and the reasons for them are portrayed, since the disruption created by them to the case study's application created the actual need for offline supported functionalities in the first place.

2.1 Web Applications

In the past decade the software industry as a whole has been facing a major paradigm shift on the way how and where applications are developed and operated. Progress in web technologies and standards have made it possible for the browser environment to become a more and more powerful but still universal computing platform. Software applications that were previously built for different kind of operating system and CPU combinations are now written with web technologies and run on the browser. This change is mainly accomplished by the power of distribution the web platform provides. Software provider or the organization's IT function does not have to take care of updating end user software to the latest version anymore, because the client-server paradigm[2] makes running outdated software version virtually impossible. In a traditional desktop application patching the software to fix bugs or introducing a new feature requires downloading a installer or a patcher and then it must be run on the computer. This requires action from the user or from the IT function. This might not cover the whole upgrading process in every situation: sometimes updating dependent libraries or operating system might be required. It also enables possibility for different computers to be running different versions of the software, creating circumstances for version mismatches between different clients. [3] [4]

The move from individual installations to a centralized services reverses the general model how software is operated, the model which has been dominating since the

start of the personal computing revolution [3, Chapter 3.3]. As stated above, this comes with many benefits. One of the most important ones is the (almost) unified platform provided for the developers. Thanks to automatic update mechanisms of the modern browsers, this platform is one of the fastest updating ones there is when it comes to speed of patches applied to end users' computers [5]. Due to the easiness of the deployment process, it is not unusual to see web application rolling in releases to production on daily or even on hourly basis [3, Chapter 3.3].

2.1.1 Single-Page Applications

Executing complicated applications on the web platform has been made possible by the evolution of the web pages from "classic", static presentation consisting of single pages to a more interactive and collaborative web applications. In the early phases of World Wide Web user navigated through hyperlinks within single pages each of them being formed and served by the server. Client, the browser, was responsible only for rendering the response of the server. [4] The current state of technology allows more of the functionality to be moved or duplicated from the server to the client. When before web pages on the browser were stateless, and the state transforms were done while moving to a new page, now it is possible for the browser to handle the state transitions and retrieve only the needed data from the server when required via AJAX[1]-requests. [6]

Applications following the paradigm described above can be generalized to be *single-page applications*. These kinds of web applications makes a request to the server, and based on the data of the server's response, they form the layout of the page manipulating the browser's *Document Object Model* in real time. This is contrary to the traditional paradigm where based on the client's request the server creates the layout for the data and sends an whole HTML page as a response, and the whole page is refreshed in result. This is also the etymology behind the name of the paradigm: there is no need for user to navigate away from the page, even when the views on the application change.

The application state handling on the browser environment and the increased amount of possibilities on the browser creates the basis for modern web development and the platform for the offline feature: the single-page application. Single-page application is tightly relevant to other hype terms such as *Web 2.0* and the nowadays partly old-fashioned term *DHTML*[2]. All single-page applications could be called Web 2.0 applications, but not all Web 2.0 applications are single-page applications. While traditional web pages where synchronously fetched and generated by the web server, single-page application relies on asynchronous pattern on data fetching. [7]

[1] Asynchronous JavaScript + XML
[2] Dynamic HTML

With the asynchronous pattern the browser can fetch resources on smaller chunks when the need arises, usually through REST APIs [8]. This also makes possible the premise for an *occasionally connected application*, meaning that the web application can function without having a continuous Internet connection [9]. The application has the ability to fetch data from the API when there is connectivity, and when the connectivity disappears it can function with application logic downloaded to the browser. This widens a lot the range of use cases which can be covered by web applications.

2.1.2 REST APIs

REST is a term first time introduced by *Roy Fielding* in his Ph.D. dissertation [10] as a description for an architecture style of networked systems. The REST acronym stands for *Representational State Transfer*. Even the term REST comes up often in materials relating to web technologies, it has to be remembered that it is more of an idea for architectural style than an actual well-defined standard.

Roy Fielding explained the term on his dissertation in the following way:

> "Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions) resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use."

The momentum behind the wide use of the REST principles rises from the key motivation behind it: REST captures the key components and ideas of which made the Web successful. These characteristics are in use with guiding the evolution of the Web today. [11]

All the single-page applications that interact with data that is used by various clients requires some kind of backend service. Usually this is provided by creating *application programming interfaces* (later on referrenced by APIs) on the backend for the client-server communication. In practice the API exposes methods for fetching and altering the server's data. Therefore APIs can be thought as the *face* for the web service, aimed for listening and responding the clients' requests. [8]

Figure 2.1 provides a presentation from Masse [8] about the general principle regarding a REST API. Well designed REST API is easily understood by humans, since the URLs themselves should describe precisely what resources will be returned to requests send to them. Using the URLs together with a describing *HTTP method* as a "verb" for the operation makes the whole design logical. A REST API can be thought as an assembly of interlinked resources. These resources forms the REST

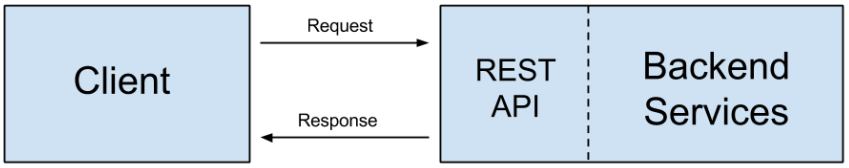

Figure 2.1: Basic structure of a REST API service, based on the work by Masse [8]
.

API's *resource model*. [8]

REST APIs plays an integral part in single-page applications. After the client side code is initialized on the browser, all the actual content (depending on the context whatever that might be) is fetched to the browser via an API which nowadays are almost always RESTful. All the modification and the creation of the data is also done through the functionality exposed to the client by the API. Usually the data is transferred between the client and the server using *JSON objects* (JavaScript Object Notation) for the payload.

2.2 Computer Supported Collaborative Work

As the automation level of work tasks keeps getting higher, many of the tasks left for human actors on the modern working life gets more complex. The nature of the work also transforms to be much more *distributed* than before, both in the sense of location and time. There is for example a lot more of problem solving and rule interpreting left for the human employees to solve which usually need to be coordinated within the other actors related to that activity. When doing decisions information from several sources have to be considered and noted. *Computer Supported Collaborative Work* (later referred to as CSCW) is a field of study concentrating on how collaborative actions and coordination of them can be aided by support of computer systems. The term was introduced for the first time in 1984 by Irene Greif and Paul M. Cashman. [12]

The Web has been seen as a prolific platform for CSCW implementations since the creation of WWW. The general idea behind WWW by Tim-Berners Lee could be seen as an implementation of a CSCW system. Despite the limitations set by the primitive web technologies of that time, the first implementations of CSCW saw daylight and wide usage already on the 1990's. Usual application for that time concentrated on storing documents on a shared workspace, where a given group could browse and share the documents relating to their work. Even with the limitations set by the web technology that would seem very primitive compared to modern standards, the power of web and "its ability to provide basic features for cooperation

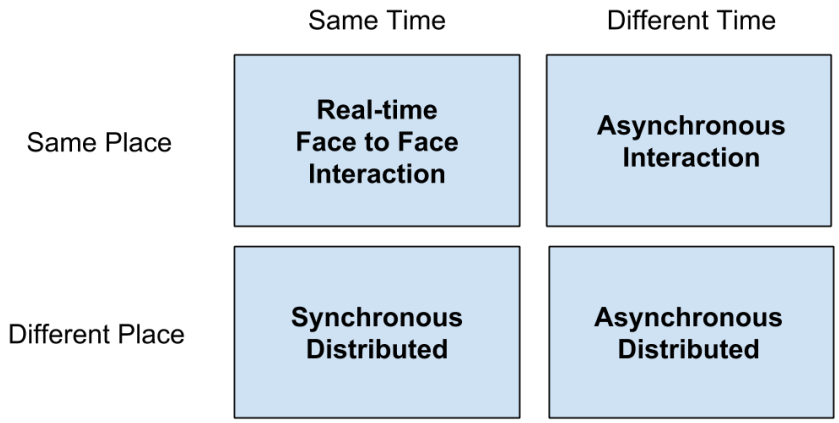

Figure 2.2: Collaborative systems in time / space matrix, based on the work by Saad and Maher [15].

in an integrated service, accessible from different computing platforms and making no demands on users to adopt new word processing, spreadsheet or other application software" was recognized to be from the highest of potentials. [13]

People using CSCW tools can often be described as a group of individuals working synchronously or asynchronously towards of achieving common goal(s). Depending on the situation they can be located on different locations geographically and/or on different time zones. In circumstances such as these all of the people involved have to have up-to-date awareness about each other's intentions, actions and results. [14]

From Figure 2.2 a reproduced time / space matrix of collaborative systems can be seen based on research done by Saad and Maher [15]. This matrix splits the types of collaborative systems in to four categories. These categories are not explicitly limited to the CSCW domain but they concern collaboration activities in general. Face to face discussion is the most common case of collaboration happening on same place and time. Traditional physical bulletin boards are an example of collaboration happening on same place different time. Video conference systems are nowadays a common scenario of people from different locations collaborating simultaneously. The evolved Internet version of bulletin boards – forums and wikis – can be seen as a way of working from different place and on a different time. That is also the sweet spot for CSCW systems; they deliver the most value to the users when they allow people to collaborate without location or time restrictions.

CSCW tools can also be used to create disruption to the hierarchies of the applied work environment, while getting more of the stakeholders involved in the creation of the information. For example in kindergartens the primary gatekeepers for the information flow towards children's homes are the kindergarten nurses. On a research

by Näsänen et al. (2009) a web service for showing parents photos taken on the kindergarten were created. One of the goals for the research was to allow efficient way of displaying the life on the kindergarten to the children's parents, whom usually just drop off and pick up the children without being otherwise present. The children also had access to smartphones with camera, allowing them to upload photos directly to the availability of homes without the moderation of kindergarten nurses. [16]

2.3 Connectivity Issues

For a single-page application it is possible to work without connection to the outside world, in case the application code is delivered to the device via some medium. For example for a spreadsheet application this might be a highly feasible idea. However for applications that interacts with the surrounding world some kind of connectivity is required in order to spare the person using the system having to input all the data into it. When it comes to CSCW applications, where one of the key points is the *collaboration*, at least occasional Internet connectivity is mandatory. This makes issues on the Internet connectivity status and quality to concern closely CSCW applications.

In many of the scenarios and application areas where CSCW is applied using desktop computers is challenging or impossible. Often the usage is outlined to the smartphone or/and tablet usage just by the requirement of having the access to the system with the users on all occasions. If the information needs to be accessed immediately on the fly, importable devices are out of the question.

For mobile devices the medium for the "last mile"[3] of connecting to the Internet is wirelessly over the radio waves using cellular network. As more and more of the usage of the Internet moves to the mobile devices, the unprecedented cellular traffic growth can easily exceed the deployment speed of cellular infrastructures [17]. This results in a certain *guaranteed unreliability* to the connections of devices using the mobile broadband as their primary method of reaching the Internet. For the time being applications where the possibility of continuous usage is critical bracing oneself to the possible temporary Internet outages is one of the factors which must be taken into account.

One possible solution in overtaking the problems caused by the excess load of cellular networks is to use Wireless LANs as the last mile. With correct configuration this makes the wireless access as reliable as is the Internet connection of the WLAN's base station. However it is not exceptional for a CSCW system to be operated on various locations, resulting in a situation where sometimes usage of cellular network

[3]Metaphor used in telecommunication industry when referring to the part of the network that reaches the customer. When looking from the customer's perspective it can be named as the "first mile."

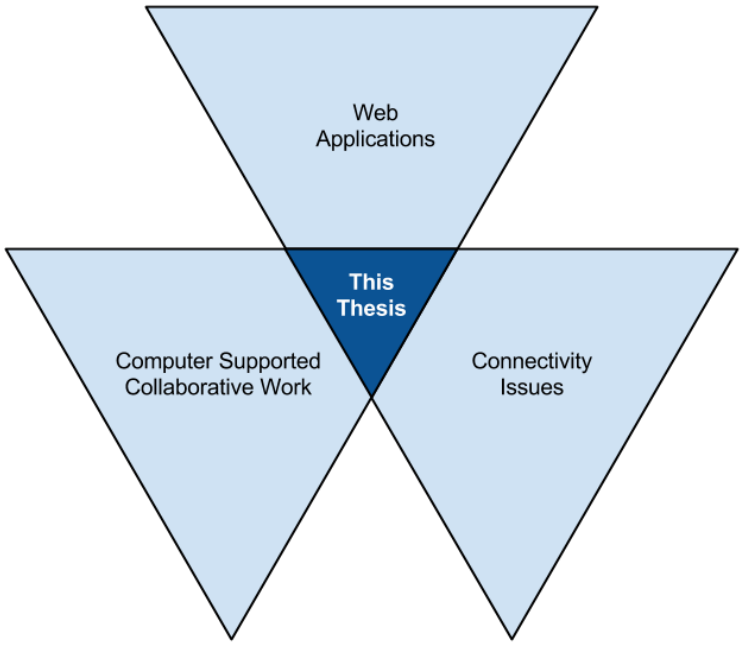

Figure 2.3: The research gap covered by this thesis

is mandatory. For applications where the SLAs[4] are strict and system is used from various locations this method does not deliver any relief.

This thesis studies the issues on the Internet connectivity only from the application level, within the possibilities provided by the browser environment on handling the problem. Possible solutions available on the network-level are ignored.

2.4 Research Gap

This thesis studies the overlap of the themes previously introduced in this chapter, as can be seen on the visualization in Figure 2.3. This thesis studies what is required from a *single-page application* which enables *computer supported collaborative work* for the users on a environment where *issues with the Internet connectivity* happens regularly.

CSCW has been studied widely for over the past two decades. The need for streamlining working methods in favor more productivity makes efficient will make applications enabling it only more essential and common.

The fast-paced evolution of web technologies have created a powerful platform for application development which is ubiquitously present and available everywhere. On the last decade it was available on desktop computers, today it reaches 40 % of the Earth's population due of increasing usage of mobile devices and tablets [18]

[4]Service Level Agreement

and by 2020 *4.9 billion* devices is estimated to be connected[5] thanks to the *Internet of Things* (IoT). This will make web technologies only more appealing method for application creation on the future.

With all this taken into account, it would not be surprising if creating applications with the web stack and the need for having an offline support on them would only get more common. Even without the rising need of these kind of applications, the user experience of any web application used over cellular network can be improved significantly with the implementation of a proper offline support, making research around it useful. On top of that there are no existing research covering all of these themes together.

[5]http://www.gartner.com/newsroom/id/2905717

3. CASE PÄIKKY

This thesis approaches the research problem through a real-life case study in which an offline support was implemented to an already existing system. This chapter describes the basics of that system and the use cases of it. The technical characteristics and domain knowledge of the system which are important for understanding the functionality of the offline support are opened up in more depth. Moreover the reasoning behind the decisions done regarding the implementation of the offline support are described.

3.1 System Description

The system under the analysis is *Päikky*, a solution for daycare personnel and daycare children's parents for planning the needed daycare and providing passage control of the children (and of the personnel) on the kindergarten.

Päikky is a collaborative web application for daycare personnel and children's parents to plan and coordinate daily activities in kindergarten. Päikky also provides functionality which helps kindergarten personnel to coordinate the daily activities on the kindergarten and communicate with the children's parents.

Päikky consists of four parts:

1. *Kindergarten UI*, mobile-first single-page application used for logging children in/out to kindergarten, referenced to as *the application* later on,
2. *Family UI*, single-page application used by parents when planning children daycare time,
3. *Manager UI*, single-page application used by kindergarten management,
4. *Backend*, Grails[1]-powered server that implements REST API and other services used by the Päikky UI's.

Under the hood *Manager UI* and *Family UI* are located in the same codebase, and from a technical point of view they are a single web application. Although they are inseparable from a user's standpoint and share the same architecture, the layout

[1]Java-based "high-productivity web application framework". http://grails.org

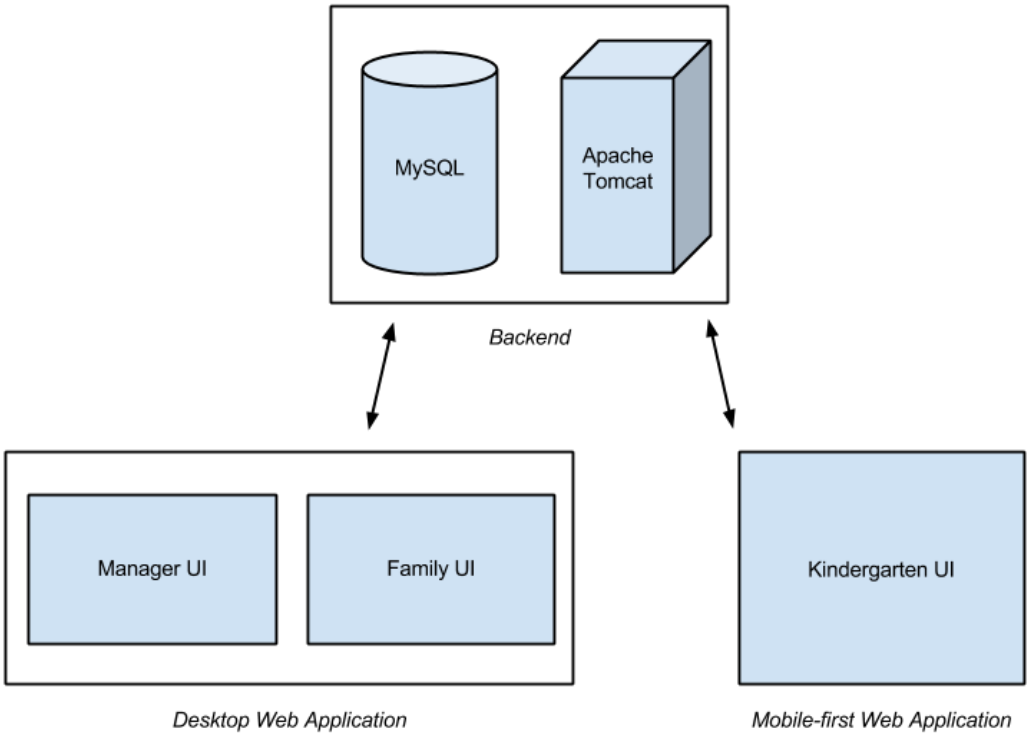

Figure 3.1: The high-level architecture of Päikky

and functionalities are completely different. Contrary to the *Kindergarten UI* they are designed to be used primarily from desktop clients.

In this thesis we are focusing mainly on the *Kindergarten UI*, which is the tool used on a daily basis by the kindergarten nurses for tracking the attendance of their care group. Some emphasis is also on the backend due to its major role on the system's overall view.

3.1.1 Backend

The backend is the web server of Päikky, which holds all the master data of the system. It serves HTML content and other resources (graphical assets, JavaScript files, Cascading Style Sheets) to the other modules of Päikky. The backend also implements a REST API which requests and alters the dat from the other modules of the system. The abilities the API offers differs between the type of the user's account, but in practice all of the data can be modified via it. Data is exchanged between the backend and the UI modules as JSON objects.

Currently the backend is operated as a single virtual machine on the Amazon Web Services cloud. The data is stored on a single *MySQL* database. The backend is multitenant (principle, where single application and database instance serves

multiple tenants): single instance serves multiple municipalities and kindergartens [19].

3.1.2 Kindergarten UI

Kindergarten UI is the part of Päikky which is used daily by the kindergarten nurses. From the Kindergarten UI nurses can also see elaborate information about the children, including allergies and persons who are allowed to pick them up, and whether parents have allowed the children to be photographed. Nurses can also easily see the contact information for each child. Updating this information is also possible.

Using the Kindergarten UI, the key activity for nurses is to log children in (and the other nurses) when they arrive at the kindergarten and log them out when a child or nurse leaves. This activity is repeated tens of times per day. To do this conveniently nurses are equipped with *Android* smart phones. The marking of goings and leavings has to be done in order to replicate the presence status from the real world to the Päikky system. Since monitoring attendance and in the future creating bills for provided daycare is based on the presence data generated by logging the personnel in and out, it is crucial that this activity can be achieved successfully under any kind of condition. Doing this activity should also be as effortless and simple as possible for the nurses so that it would get done at the exact moment when the actual event happens in the kindergarten.

Currently in Finland families are charged from the kindergarten service on a fixed price basis. Every child with similar care plans is charged with an equal amount. Upcoming changes in Finnish daycare legislation changes the basis for payment to be hour rate based, which creates the need for systems like Päikky that are able to track the children's attendance on the kindergarten for billing purposes. This makes the accuracy of the attendance data to be critical for the success of the system as a whole.

The simplicity requirement goes for every other aspect of the system: the Päikky users' demography is a very mixed crowd. The kindergarten nurses' age can be anything from 18 to 65. Because of the age variation also the ability and the starting level to use a digital service via smartphone varies a lot. Taking the easiness of usage is also one of the key principles behind Päikky's user interface and interaction design. It is also one of the unique selling points of the company behind Päikky, the *MukavaIT*[2]: *"using IT systems should not be hard or unpleasant"*.

Based on the plans done by parents on the *Family UI*, nurses can see how many and who of the children they are expecting to appear for each day. If the parents

[2]http://mukavait.fi

have to change the already existing plans with short notice, automated message is sent to the nurses stating the change, and the plans visible for the child relating the case get updated in real time. Nurses can also see the exact amount of children and nurses present at the kindergarten for any given time.

As stated above, the attendance of nurses can also be tracked with the Kindergarten UI. With the *Manager UI* it is also possible to plan shifts for the nurses. Combined with the planned attendance data of children done by the parents, this makes Päikky a powerful tool for organizing the kindergarten's daily schedule and ensuring that there are always enough nurses present to take care on the children. This is important since required nurses/children ratio is dictated by law in Finland.

From technical point of view looking the Kindergarten UI is a single-page web application created to be used primarily with mobile devices. The libraries the application consists of are the following:

1. *Backbone*[3], *Model-View-Template* –framework providing the skeleton for the application,
2. *Marionette*[4], library of common design and implementation patterns for Backbone,
3. *RequireJS*[5], module loader and dependency manager,
4. *Underscore*[6], functional programming inspired library for utility functions,
5. *jQuery*[7], utility library for DOM manipulation,
6. *Moment*[8], time and timezone handling library.

The method of data synchronizing between Kindergarten UI clients and the backend is *polling*. Each client sends checksums of its attendance data to the backend on a regular interval, and if the backend calculates different checksum for the data requested, new data is returned to the clients. This means that if child is logged in to kindergarten with device A, it can take as long as the configured interval for the device B to receive that information. The current interval is 45 seconds. Previously the interval was only 20 seconds, but as the number of users increased the current infrastructure on which Päikky is run could not handle the amount of requests invoked in that interval.

[3]http://backbonejs.org/
[4]http://marionettejs.com/
[5]http://requirejs.org/
[6]http://underscorejs.org/
[7]http://jquery.com/
[8]http://momentjs.com/

The devices and browsers targeted during the development process and which were delivered to the kindergartens by the client organization were *Samsung Ace 3 Style*'s and the latest stable version of the *Chrome* browser. The devices use mobile broadband connection as their access method in reaching the Internet. Usually each of the kindergarten's care groups have at least one dedicated device at their disposal. Large groups might have two devices.

The usual – and most of the time the only – usage environment for Kindergarten UI are the kindergartens around Finland which are using Päikky. The application is used in both outdoors and indoors. Kindergartens can be located practically anywhere, and the mobile reception can vary a lot between different kindergartens. In some locations the coverage and experienced connection quality can be at par with physical broadbands, but the locations with the worst reception are very challenging when looking from the connection speed and latency point of view. It is not unusual for the kindergarten to be located on a remote location, where getting 3G connection is more of an exception than standard behaviour.

3.1.3 Presence Model

As the primary functionality for Päikky is the monitoring and planning of the attendance of the people on the kindergarten, one of the most essential data models is also the one implementing this feature. The backbone for holding and handling this data is Päikky's *presence model*. In brief presence model is an entity which indicates a range of time when person has had a certain status. These entities are persisted by the Päikky backend in the database.

There are different *purposes* for presences: *actual, plan change, plan* and *default plan*. Each presence has always a single purpose. The purposes are similar for both children and nurses. Each of the purposes exists on their own domain, resulting in a layered construct of presences for each person. The *actual* presences are the most significant, while the *default plan* is the least significant. This is visualized in Figure 3.2. Person's presences with same purpose may never overlap. For single person there can be only one or zero presences per purpose at any given time.

Each presence also has a single type. The ones used the most are *present, sick,* and *day off*. These basic types are the same for children and nurses, but nurses also have a extended set of types indicating specialized reasons for not being at the kindergarten. For example these types includes likes of different cases of sick leaves, trainings and holidays. In this thesis we are concentrating at looking the system primarily from the children's presence markings perspective.

A single presence always belongs to a single person. The presence always has a start time and an end time. There is one special case when presence does not need to have an end time (it is then set null in the database): if the purpose for presence is

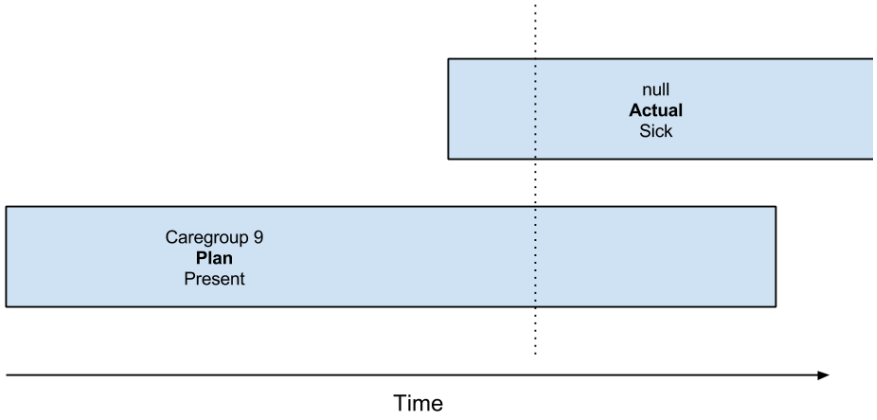

Figure 3.2: The presence model of Päikky.

actual and that presence is the latest ongoing presence for the person. For example when a person is logged in to the daycare in the morning, a new presence entity is created. For this presence start time is set to be the time when he/she arrived to the kindergarten, and end time is left null. This means that this presence is *active* for the current person, indicating ongoing activity. When the person leaves the kindergarten the active presence's end time is set to be the leaving time. If their state is otherwise altered, the active presence is ended and new presence entity with the new type is created. The ending time for the previous presence is the same as the starting time for the new presence.

3.1.4 Presence Status and Presence State Machine

Based on the presence entities of an individual person a *presence status* for them can be determined at any time. For example if person has a planned presence for the day but they has not arrived at the kindergarten yet, the status would be transcribed as "not yet present" indicating that this person will be present later on today. Mutually if the person has been today at the kindergarten but has already left, their status would be "left for today". The status mechanism is implemented for providing more information about the current attendance status for the kindergarten personnel, in contrary to what simple boolean "present / away" statuses would implicate.

Within presences sharing the same presence purpose, there is a set of rules of allowed state transitions. These rules can be visualized as a *finite-state machine* that shows the possible transitions from different status into another.

Figure 3.3 describes a simplified version of Päikky's state machine. In this finite-state machine each state indicates two things: is there an active ongoing presence entity for the individual person which has "actual" as the presence purpose ("inactive / active" on the visualization), and if the current state means that the person

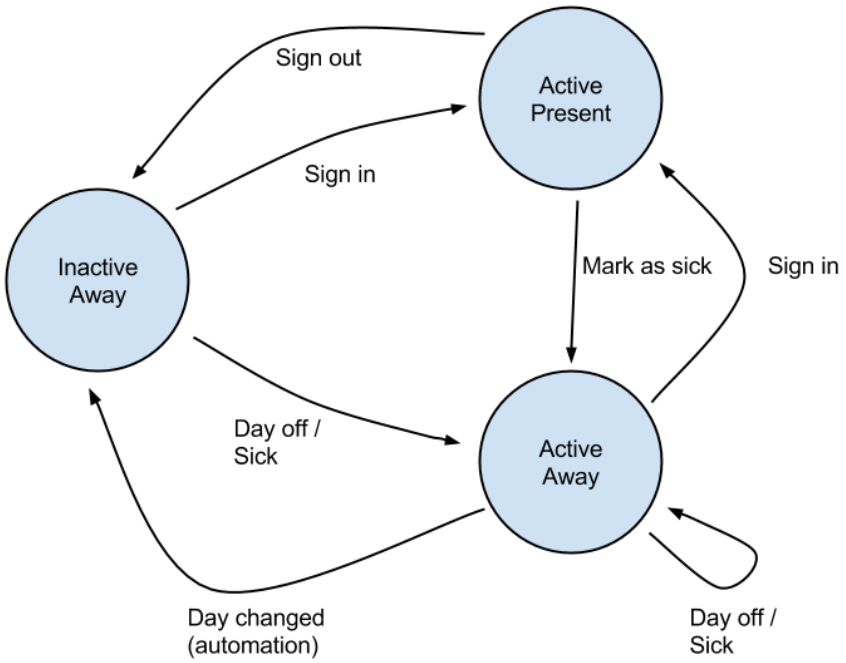

Figure 3.3: Presence State Machine. For visualizing purposes this state machine is a simplified version of the one actually implemented in Päikky

is physically present at the kindergarten or not ("present / away").

Based on the allowed state transitions different kinds of buttons for altering the presence state are shown on the person's profile UI. For the most straightforward example if the person is already present on the kindergarten, only a *sign out* button is shown. Since signing in a person already present on the kindergarten is prohibited by the presence state machine, the button *sign in* is therefore also hidden from the UI. This can be seen in Figure 3.4.

3.2 Need for Offline Support on Päikky

Päikky's development was originally started with creating an *MVP version*[9] of the product which was used in validating the business case of the idea. Therefore it only had a thin feature set, covering only the most essential features needed for the application. According to the client organization's CEO the need for some level of of offline support was noticed at an early phase of the system's development. The marking of persons' status was also recognized to be the key activity already at the start of the development.

First time the Kindergarten UI's total dependency on the backend realized to be problematic was during a roll-out of Päikky to a kindergarten where for the first

[9]Minimum Viable Product

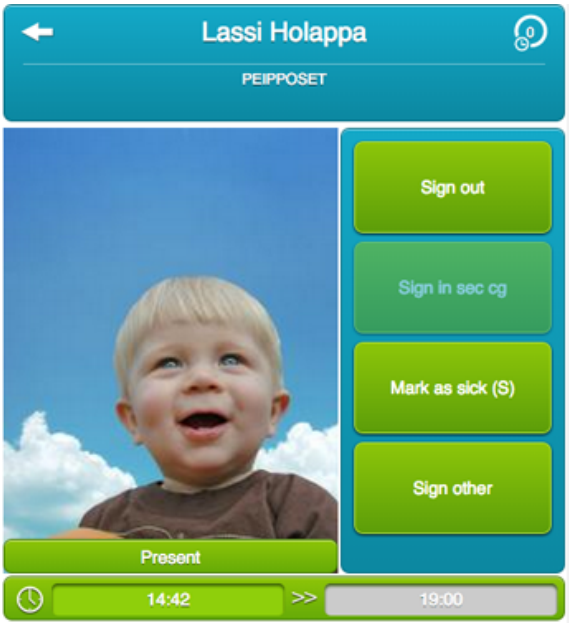

Figure 3.4: Screen capture from a child's profile on the Kindergarten UI

time the network quality was way worse than the average. On that site the latency experienced on the Internet connection was often couple of seconds. This caused the changing of the state of a person on Päikky to be frustratingly slow, since the new state would update only after the server response was received. Although this did not yet form to be a obligatory need for the offline support, since it could have been solved with a more lighter solution.

The start for the process of planning and implementing the offline support and was a *request for tender* in which the client organization participated. On this request for tender it was explicitly stated the requirement for logging of persons to be possible under any kind of network condition. Fulfilling this requirement was also seen as a good prospect for improving the user experience of the application as a whole.

The offline support would have materialized to the product even without the existence of the request for tender, but on that scenario the possible schedule for it remains unknown. It was also speculated that on that scenario the level of offline support provided could have been slightly lighter.

4. RESEARCH METHODS

This chapter introduces the methods followed through this thesis and describes how they are applied on the research.

4.1 Design Science Research

This thesis studies the research questions by adapting methodologies and toolbox provided *design science research* (DSR). On the literature this research framework is also referred to *design science* and *design research*.

Hevner and Chatterjee [20] define design science research as follows:

> "Design science research is a research paradigm in which a designer answers questions relevant to human problems via the creation of innovative artifacts, thereby contributing new knowledge to the body of scientific evidence. The designed artifacts are both useful and fundamental in understanding that problem."

Usually when DSR is applied three related cycles of operations can be recognized, as can be seen in Figure 4.1 [21]. First there is the *relevance cycle*, which works as an interface towards the application domain and gathers the information required for producing the artifact. Secondly there is the *rigor cycle*, which provides the existing scientific theories and methods to aid the creation of the artifact. The rigor cycle works both ways, providing feedback on the theories applied on the creation of the artifact and growing up the knowledge base by adding the results to the academic continuum. Finally there is the *design cycle*, where the information from both the application domain and knowledge base meets and possible artifacts for solving the problems found on the application domain are built and evaluated.

The goal for DSR is that through the three cycles described above, solutions for real-world business problems are found. As a by-product new insights found are increasing the size of the knowledge base via the feedback to the rigor cycle, while the resulting artifacts can be implemented to the application domain via feedback to the relevance cycle. [22]

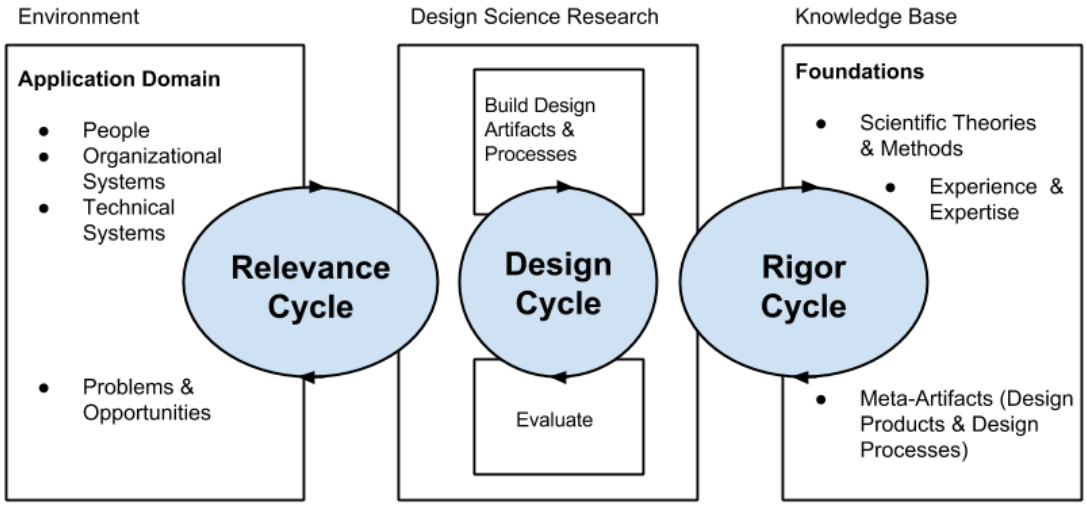

Figure 4.1: Design Science Research Cycles, based on the work by Hevner [21]

4.2 Case Study

Usually in any research, no matter in which field it is conducted, the size of the sample is directly linked to the quality of the research's outcome. Involving large numbers of participants to the study results in a broader and more representative sample. In some studies research done with a smaller sample will not necessarily be statistically as clear as using a large sample would be. [23, page 144]

Getting a large sample can be extremely challenging or even impossible on some occasions. This should not be a barrier forbidding the conducting of the research, since methods for getting valid results with smaller sample sizes also exists. One of these methods is *case study*, which is also one of the methods applied within this thesis. [23, page 144]

It is also commonly questioned that how valid base does a single case study provide for a scientific generalization. There is no straightforward answer for that. To put it shortly, the aim for case studies should be *"to expand and generalize theories (analytic generalization)"*, not *"to enumerate frequencies (statistical generalization)"*. [24, page 15]

On this thesis the research problem is studied in the context of a *single case study* using methods of *design science research*. The possible solutions found and implemented in the case are then evaluated with *semi-structured interviews*.

4.3 Data Collection with Semi-Structured Interviews

This section covers the methods on collection of the data about the user experience on the offline support done by conduction series of semi-structured interviews.

As stated in the previous chapters, there is no such thing as an average user of Päikky, due to the fact that users of the system have very different kind of backgrounds. The preparedness for using of technical devices might be very good or it can be almost nonexistent. Because of that one of the goals for this thesis was to study if the offline support implemented is actually understood by the users of Päikky. The received user experience is studied through a series of interviews executed with the nurses using Päikky.

In addition also a interview with the CEO[1] of Päikky was performed. The goal for the CEO's interview was to find out the reasoning and the motives for the offline support development. The amount of available resources and the reason of the allocating them the way which was done is also discussed. The results of the CEO interview are documented thorough this thesis on situations where references to the client organization's decisions can be found and especially in Section 3.2. Otherwise this section focuses on the interviews done with the kindergarten personnel.

The structure used on the Päikky nurses' interviews can be found from Appendix A.

4.3.1 Finding Interviewees

Aim for the interviews was to find interviewees whom have used Päikky for an extended period of time, and if possible, also before the production roll-out of the offline support of the system. Other notable factor for the interviewees was the location; it would not been prolific to interview nurses on kindergartens where mobile broadband coverage would not have any issues. On locations where solid Internet connectivity is a given it could be possible that the users would not even have notice the existence of the offline support.

Since Päikky is a product sold explicitly by the client organization, the obvious mean of getting interviewee candidate was to go through them. The client organization provided contact information of two kindergarten managers, whom geographical location had issues with the mobile broadband coverage and the time frame of Päikky's usage would match the requirements set for the interview. These managers were then approached and asked if there would be any volunteers on their staff to take part in a 45 minute interview done on the kindergarten's premises. It was noted to the managers that the optimum interviewee candidate would have used Päikky prior and after the implementation of the offline support, and they would be

[1] Chief Executive Officer

within the ones who are using Päikky on daily basis.

This process resulted in four interviewees being found, two from each of the kindergartens.

4.3.2 Preparing Interviews

Prior to the actual execution of the interviews a semi-structured format for the interview was created. The purpose for this structure was to offer an baseline and a general plan for the interview.

Instead of having a strict, hard-coded script for the interview, the structure was meant to aid the interview process to find out answers and issues that were possibly not realized by the interviewer before actually finding them out during the interview. The questions were meant to set the topic and then allow the interviewer to dig out all the possible aspects from that topic.

4.3.3 Conducting Interviews

Four kindergarten nurses were interviewed in total. All of the interviews were conducted face-to-face locally on their kindergarten.

Every interview started with asking the basic information from the interviewee. Any directly identifying information were not collected. These questions could be considered as a warm-up for the more important topics, but they also were on a key spot providing background information about the interviewee to the interviewer.

The middle part of the interviews concentrated about how the interviewees used Päikky, and which kind of issues they had faced on it. Every time they described an issue, they were asked how they handled it.

After that the interviewees were asked about their usage of the offline mode on the Kindergarten UI. How often had they experienced the application going into offline mode, how did it change their usage of Päikky, and what kind of issues had they experienced. Also the influence about the limited feature set of offline mode is investigated: did it impede the usage of the application and if so, how.

Also the understanding of the concepts related to the offline mode were enquired: what they considered happening when the device entered the offline mode, how did they understand the under-the-hood events when the device came back from the offline mode. Also the impressions about the location of the data were asked. No direct references were made to the client-server -architecture, but one of the topics aimed to determine if the interviewees had any idea about the master data not being located on their smartphones.

At the end of the interview the interviewees were asked to have a free word on any topic considering Päikky – the good parts, the bad parts, and the features they

wished would exist on Päikky.

Since Finnish was the native language for the interviewer and all the interviewees, the interviews were conducted in that language.

In all the questions were interviewee's usage of Päikky was asked, the question was aimed to be put in the form of "describe me last time you executed the [action] with Päikky". This was done trying to get the interviewee to reminisce the actual usage of Päikky, instead of setting them mentally trying to guess what would be the optimal and the instructed way (by the client organization's trainings) of conducting the asked action on Päikky.

During the interview only the interviewer and the interviewee were present. No notes were made during the interview, the situation was aimed to have a relaxed athmosphere, more of a conversation instead of an "official interview." Interviewees were told that if some of the questions seemed too distracting, it would be OK to skip any of them they wish.

All the interviews were recorded. It was also stated that they would not be identifiable from the results presented on this thesis. After this thesis is finalized, it was promised to the interviewees that the recordings would get destroyed.

4.3.4 Analyzing Interviews

After the interviews the recordings were listened thoroughly and littered into text. The interview recordings were not exported to the textual format from word for word, but instead key ideas and themes were deducted. Due to the conversational nature of the interviews, word for word littering would not have been beneficial to the research.

After the major themes for each individual interviewee were found, they were cross-matched in order to find what perspectives were mutual for each of the interview. Also individual interesting viewpoints opinions were noted on the analyzing process.

The interpreted results of the interviews are presented later on the Chapter 6.

5. IMPLEMENTATION

This chapter coverages what was done to the Päikky system in order to get it working also during the Internet connection shortages, and how that was done.

This chapter is concerned with the facts of what the offline mode consists of and how it was implemented. Reasoning for the decisions made are also explained. Analyzing the consequences of those decisions are done later on Chapter 6.

5.1 The Goal for the Offline Support

The primary goal for the offline support on Päikky was to enable the most critical tasks for kindergarten nurses on all situations on the Kindergarten UI (the mobile friendly version of the Päikky system).

Päikky's main and the most important feature is the ability to track the attendance of the kindergarten's children on real time. In order to achieve that, nurses must be able to mark the children's coming and going without getting interrupted by the limitations of the application. In order to offer offline support on the Kindergarten UI, a new feature called *Offline mode* was implemented. Offline mode aims at removing the Internet connection quality related limitations on the nurses' key activity. Nurses should be able to use the children logging feature on Päikky under any kind of Internet condition. If the Internet connection is nonexistent, the application should record user's actions and save them to the server once the Internet connection is achieved again.

The secondary goal for the offline support were to allow as seamless usage as possible of the Päikky on kindergartens where Internet connection is weak. Nurses should be able to see the information from Päikky even if there is no Internet connection at the time. Information should be served based on the best-effort delivery: application should show all the information it has at the time to the user while trying to fetch the most latest version of the information.

Other parts of Päikky, the *Manager UI* and the *Family UI*, were left out of the offline support.

The ultimate vision – yet to be reached – for the offline support is that it would abstract the Internet connection quality completely away from the user's consideration. Under any connection quality the user should be able to use Päikky normally without any interference. In the current version this is not yet achieved (nor was it

scoped to be achieved).

5.2 State Transitions to and from the Offline Mode

Implementation of the offline support to the Kindergarten UI made it to have two different states related to the Internet connection status. Application is on the already mentioned *Offline mode* when the Internet connection is poor or nonexistent, and on the *Online mode* when the Internet connection is working normally.

The current mode is implicated to the user clearly: while in Offline mode, the Kindergarten UI's header changes color scheme to greyscale and the title says directly *"You are working on the offline mode"*, localized to the user's language.

While in Online mode the Kindergarten UI works almost identically as it did before the offline support implementation. Major changes are that all the API requests done to the Päikky backend are cached to the Local Storage of the device running the Kindergarten UI. Also the sending of Presence marking changes to the backend are done by a dedicated component. Both of these are primarily refactoring the inner parts of the Kindergarten UI, while using the application the user should not experience any difference to the versions prior from these changes.

When entering the Offline mode the method on how data is fetched changes. Instead of fetching the data user requests from the backend, the Kindergarten UI fall backs to the data cached on the *Local Storage*. This will not guarantee the availability of the up-to-date information to the user, but at least showing the best effort version is possible. Due to the nature of the Päikky's data, in most of the cases this is acceptable (for example when looking for children's parents' phone numbers, and other similar data that is not altered on daily basis). Also some of the features on the Kindergarten UI are disabled when the Offline mode comes active.

While the Offline mode is active, the component responsible for sending the Presence marking changes – the *Job Queue* – also acts differently. If there is a recognized issue with the Internet connectivity (which triggers the Offline mode to be activated), Job Queue stops sending the Presence Markings to the backend but instead saves them to the Local Storage. User is notified on this by showing all the time the size of the queue on the top right corner of the Kindergarten UI. When the Internet connection is active again, the Job Queue starts to send the cached Presence Markings to the backend one at a time.

5.3 Limited Feature Set in Offline Mode

Similarly to many other real life software project, also in Päikky compromises have been made while balancing between the scope, available resources, and the quality of the end product. In order to create software with good quality under given budget,

the scope had to be kept reasonable and some prioritization between features had to be made. This resulted the first version (the one studied by this thesis) to be technically quite simple and even naive on some aspects. Users experience this as a lack or disabling of some features on the offline mode.

5.3.1 Disabled Features

When the application enters the Offline mode all the features except the critical key functionality are disabled from the user. These include features such as

1. sending messages in the application,
2. editing persons' information,
3. changing persons' photos,
4. editing existing *presence markings* or upcoming *presence plans*.

These features were left out from the Offline support based on feature importance evaluation done by the client organization.

None of the listed activities are essential for the Päikky's key feature, the real time tracking of kindergarten's personnel attendance. If there is no Internet connection available at the time, each one of these activities can be postponed without sacrificing the integrity of the presence data on the system until the Internet connection is available again.

5.3.2 Simplified Data Synchronizing

The nature of the Päikky usage by the nurses allowed the development team to make some simplifications to the implementation of the offline mode. These simplifications included the way how conflicts between concurrent changes are solved. To put it bluntly, they are not solved in any way.

To understand why this solution was feasible, one has to understand the environment and practices about how Päikky is operated. The usual scenario in kindergartens where Päikky is used is that each kindergarten group has only one dedicated device. With minor exceptions all of the presence markings for the care group's personnel are done via the group's dedicated device, which is also the only device for the group to access Päikky. This is also the way how usage of Päikky is instructed by the client organization to the kindergarten personnel. Editing person's presence status from different devices while on offline mode is especially discouraged. With these guidelines the probability for cases where two different devices have made concurrent changes to individual person becomes almost non-existent.

These circumstances allowed the development team to streamline the data synchronization on the Päikky server. On agreement with the client organization, there is no functionality that tries to solve possible merge conflicts on the presence data. If there appears a situation where two devices have concurrently changed the attendance status of a single person – contrary on the instructions given to the users – both of the changes are saved to the database. Fixing the data to reflect the situation happened in the real world is left to the responsibility of the kindergarten personnel.

The decision which passed on the responsibility of the data correctness to users also removed functionality needed on the Päikky backend. Because of this the work done in order to implement the required level of offline support to Päikky was almost entirely done to the Kindergarten UI. The backend required only minimal changes relating this. The backend changes that were required are explained in depth in Subsection 5.4.6.

The decision of implementing no merge conflict solving strategy removes the need of offline related functionality on the Päikky server almost completely. By this the development effort was cut significantly: the probability of creating data conflicts has been made minimal with the user instructions, and in the implausible case of a conflict to appear resolving it is left to the responsibility of the user, not by the code base.

From the server point of view, presence markings done on the offline mode are received and stored exactly same way as are the markings done real time on the online mode. Only thing that differs is the time gap between the presence marking's timestamp and the occasion when the presence marking is received on the server.

5.4 Technical Details

The technical details of the implementation cover almost entirely only the Kindergarten UI of the Päikky system. This is due to the allowed boundaries for the development team and the real life limitations of the Päikky system usage. In order to achieve the required level of offline support on the system, almost all of the work could have been done only on the mobile frontend codebase: the Kindergarten UI. In addition to the changes made to the backend, no other modules (*Family UI, Manager UI*) of the system needed changes. However the other modules did not gain any kind of added offline support either.

5.4.1 HTTP Cache Headers

Starting point for the offline support implementation was to take as much advantage as possible from the techniques already in use on the Päikky system's implementa-

tion. The first task for the development team was to ensure that the cache related features of the HTTP protocol were utilized thoroughly.

Previously there were issues reported by the users after production environment updates that the new features announced were not visible on their devices. This was the result of poor and some part nonexistent cache header usage. The cache headers prior the change instructed the browser to save the *index.html* document – the starting point of the Kindergarten UI web application – and all the JavaScript files for 24 hours on the file system of the device. If those files were requested within that 24 hours, the cached versions were used. This created a huge lag on the rollout of the latest version to the end users' devices, which caused work for the development team since both the old and the new version of the frontend client had to be supported on the backend.

The "release lag" was addressed by altering the HTTP 1.1 headers related to caching, which are returned by the Päikky backend's Apache web server to the browser. The first solution was to disable cache entirely, forcing the browser to fetch the data again each time from the backend while browsing. This was achieved via headers described in Listing 5.1.

Listing 5.1: Päikky HTTP Cache Headers

```
Cache-Control:max-age=0, no-cache, no-store, must-revalidate
Pragma:no-cache
Date:Mon, 24 Nov 2014 08:36:04 GMT
Expires:Wed, 11 Jan 1984 05:00:00 GMT
```

The goal with these headers is that the browser saves intentionally none of the content it receives. Achieving this is done by several mechanisms to cover the most of the browsers in use. From the HTTP 1.1 protocol view of point some of the instructions overlap each other (for example `Pragma:no-cache` and `Cache-Control:no-cache`) while some are there to address the HTTP 1.0 protocol (using `Expire-date` from the past). [25]

Using these headers solved the release lag problem and streamlined the production deploy process, but otherwise did little to actually help the development team to get closer to the offline support on the system. Actually these changes made the Kindergarten UI to be more reliable on Internet connection, since browsers were instructed not to save any data fetched from the Päikky backend. This also increased the average loads of the backend.

To take advantage of the HTTP Cache Headers from the offline support perspective, different cache rules were made for the most bandwidth-greedy assets: the images. On the backend, the following headers were set to be sent on response to each request that were made for images, no matter if they were graphical assets to the web application (icons etc) or profile pictures of children and nurses:

Listing 5.2: Päikky HTTP Cache Headers for images

```
Cache-Control:max-age=86400
Content-Type:image/jpeg
Date:Mon, 24 Nov 2014 12:41:01 GMT
Expires:Tue, 25 Nov 2014 12:41:01 GMT
```

These headers allow the browser to cache the image for 24 hours. For person profile pictures there will not be any lag on updates when the image changes. Every picture gets a generated unique file name on the upload, so from the browser's point of view they are totally new pictures instead of new version of the old picture. The cache validness duration can therefore be decided based only on how fast the update cycle on the graphical assets of the needs to be. If generating unique names for the graphical assets would be done on the production version build phase, this cache duration could be increased to for example one year and only side effect would be increased cache sizes on the clients' browsers.

Addressing the HTTP cache header related problems were not directly coupled to the enabling of the offline mode, but fixing them would have benefit the Päikky system even if the offline support would not have been on the product's road map.

5.4.2 Application Cache

HTML 5 specification adds a new tool aiding the creation of offline supported web applications: the Application Cache. The specification was designed to allow creating of web applications that would work (after caching) without Internet connection, but it is also useful in decreasing load and start up times in a normal, Internet-aided usage. [26]

Application cache is initialized by creating a Cache Manifest file for each HTML document. In Single-Page Applications, which Päikky also is, this is achieved easily since as the name implies there is only a single HTTP document for the whole web application that needs to be loaded [27]. Whereas HTTP headers regarding cache rules addresses the problem with generic information about the expiry date of the fetched resources, cache manifest states more elaborate instructions for situations where Internet connection is unavailable. [17]

Listing 5.3: Snippet of Päikky's Cache Manifest

```
CACHE:
js/configurations/local.js
js/main-b13f3e6.js
img/icon_delete.png
img/paikky_logo.png
css/client.css

NETWORK:
*
```

In Listing 5.3 a shortened version of the Cache Manifest of Päikky's *index.html* can be seen. The manifest describes the nature of the applications resources regarding network status to the browser. On the complete version of the Päikky's manifest each of the known resources for the application is listed under the ''CACHE:'' notation, which means that these resources should be cached by the browser. After that everything else is whitelisted with the ''*'' wild card to be downloaded from the network. The whitelisting of the rest of the possible resources is important, since when working with the Cache Manifest browser tries to do exactly as stated on the manifest. This means that if that wild card network rule would be removed, the browser would not be able to fetch the resources not listed on explicitly stated on the manifest. [28, page 212]

When using the Cache Manifest, it must be noted that even when the user is online, the files are served from the Application Cache. Developers must also notice that updating the resources listed on the Cache Manifest is not enough. The browser does not re-download them if the manifest itself is not updated (or if the cache expiry date set by the HTTP headers for individual resource does not expire). When visiting a site with a cache manifest again, the browser renders the first version of the site with resources stored on the Application Cache, and only after that connects to Internet for checking possible updates to the content. [29]

Due to the nature of the cache manifest (has to be updated on every web application update, has to handle each of the resources somehow) it is highly recommended that the creation of the manifest file is automated as a part of the build process of the web application. In Päikky's Kindergarten UI, this is done with the *grunt-manifest* tool every time a new production version is created. For the development environments, a different, static cache manifest is used, which tells the browser not to use the application cache at all.

It is also possible to insert a fallback version for the resources on the Cache Manifest that are used if the network connection is unavailable. In practice the manifest can state alternate version for resources to be used, if there is no network connectivity. This can be useful in some browser heavy applications that use minimal

amount of data from the Internet, for example applications like *Google Spreadsheet*[1] [30]. However this was not seen useful on the implementation of Päikky's offline support and therefore was not used.

As was the case with the HTTP Cache Headers, the Cache Manifest would also have benefit Päikky even if the offline support would not have been implemented to the system.

5.4.3 Monitoring Connection Quality

Identifying connection issues is a key activity regarding the Offline mode. When a connection issue is noticed, the Kindergarten UI fall backs from the Online mode to the Offline mode. While on the Offline mode, the application must monitor the health of the connection to determine when it is safe to use the network connection again.

On Päikky the connection quality is monitored via errors on *Ajax requests* [31], triggered by the *jQuery*-library. As described before, the Kindergarten UI communicates to the backend via REST API requests for different kind of resources. There is an application level error handler for cases when one of these requests should fail. Main part of this handler can be seen in Listing 5.4.

Listing 5.4: Code Snippet which triggers Päikky's Offline Mode

```
// Function for identifying network related Ajax errors
var isOfflineTriggerError = function(thrownError) {
    return _.contains([
        'timeout', // "JavaScript-level" timeout
        ''         // "network-level" errors - all of them
    ], thrownError);
};

// Error handler for Ajax errors
$(document).on("ajaxError",
    function (e, jqXHR, ajaxSettings, thrownError) {
    // List of exceptions that triggers offline-mode
    if (networkStatus.isOfflineTriggerError(thrownError)) {
        vent.trigger('offline');
    }
});
```

In practice Kindergarten UI triggers Offline mode on two different kind of errors: if there is a error coming from the JavaScript-level (usually meaning that no response for request was received within the configured time period) or if there was an error on the network level (connectivity was nonexistent). In these cases, an *offline-event*

[1] http://spreadsheet.google.com/

is triggered via the *vent*, the event bus of the application.

Modern browsers also implement the *navigator.onLine* property, which indicates the network status as a boolean value. Despite the promising specification this can not be the only way of checking the network connectivity, since the behaviour is heavily dependent on the browser. For example in *Chrome* and *Safari* having only a connectivity to local area network will result the *onLine* property being true. [32]

After the application has entered the Offline mode, it halts all the Ajax requests except a specified *ping*-request to the backend which is used to determine the health of the connection. If the ping should fail, another ping request is scheduled. There is a maximum time interval configured on the application in which the response must be received. Only after a successful ping request within the allowed time range the pinging is stopped and the Online mode is activated again.

5.4.4 Using Local Storage as a Cache

The new HTML5 standard adds also another important tool for the Päikky's offline support to take benefit from: the *Local Storage* [33]. It is an origin specific storage which saves its contents to the user's file system and persists them between sessions. While being only a simple key-value -storage it is not the most sophisticated solution for client-side storage mechanisms there is available on the modern web development toolbox, but for simple use cases like the Kindergarten UI its simplicity is a decisive advantage.

While being on the Online mode, every request made by Kindergarten UI to the backend's API is stored to the Local Storage. The request's URL is used as the key for the entry, and the response of request is saved as a value for that key. At the current version of the application there are no optimization mechanisms implemented with the Local Storage content. When Internet connection is available the saved responses are ignored. On each request, the possibly already existing responses are overwritten.

After the Offline mode is activated the Kindergarten UI continues to make API requests based on user's actions. But instead of making them to the backend's API, the *Backbone*-library – which is responsible for data fetching and synchronizing on the application – forwards the requests to the Local Storage API, where cached versions of the once fetched requests can be found. This allows the user to navigate within the application and see the latest version he/she has fetched from the backend. If the user navigates to a view not visited before, view without the actual content is rendered.

Due to the limited feature set of the Offline mode there is no need for checking dirty cache entries [34, page 138], since user can not alter the cached data while having no Internet connectivity. Only after the Online mode is activated the editing

of the data is made possible, and then the contents of the cache is ignored. Exception to this are the *Presence Markings*, whose are explained in more detail on the following section.

5.4.5 Job Queue: Promise-based Presence Marking Queue

The key activity for nurses recognized before on this thesis is the altering of the kindergarten's personnel presence data. This activity must be available for user no matter what the Internet connection status is. Allowing this activity was also the primary goal for the offline support to have. With the resource limitations the development process had it was therefore apposite to create a dedicated mechanism for syncing specifically this data and disabling other data altering features while the Offline mode is active.

In the Kindergarten UI, this is enabled by a component called the *Job Queue.* Job Queue is an array of jQuery's *Promise* objects. Each objects in the array represents a single Presence Marking to be saved to the backend. Every time the user creates a new Presence Marking, it is added to the queue, no matter if there is an Internet connection problem recognized or not.

The concept of "promise" was first time introduced in 1976 by Friedman and Wise [35]. It is used to describe an object whose value is yet to be resolved, being a temporary placeholder and a *promise* for the to-be value. In asynchronous languages such as JavaScript this is highly useful since for example when fetching data the developer can not trust that the results are available immediately after the request. Promise makes sure that altering the state or the progress of its internal request is not possible from outside. This makes it an ideal capsule to wrap the AJAX requests with. [36]

When the user creates a new Presence Marking on the Kindergarten UI (for example logs child in to the kindergarten when he/she arrives), a new Promise is created which includes Ajax request to the backend. Request's payload is the new Presence Marking. That Promise is then added to the job queue. In Figure 5.1 a screenshot of the Kindergarten UI is given while the offline mode is active. In the figure, the size of the job queue is indicated to be *five*.

While the Online mode is active, job queue constantly looks for the oldest job on the queue and tries to execute it. Executing it means activating the Ajax request inside the Promise. After the request is successfully sent to the backend and a response is received, the promise marks itself as *solved* and that job is then removed from the queue. This process is repeated within a configured interval as long as the application is active on user's browser. If an ongoing request fails due to a network error, the application fall backs to the Offline mode, and the job queue execution halts until Internet connectivity is gained again. It is possible for the request also

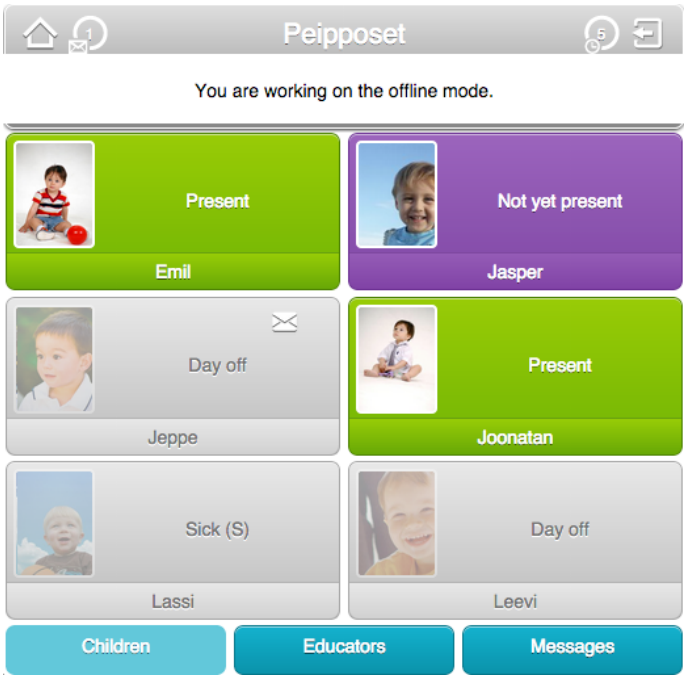

Figure 5.1: Kindergarten UI's care group view, offline mode being active. The size of the presence marking queue is indicated at the top right corner of the user interface.

to fail due a mismatched parameters, which causes the job being removed from the queue.

The state of the queue is saved on the local storage to address the case where the user shuts down the application before the whole queue is sent to the server. Therefore every time the application starts the queue is initialized from the contents stored locally on the user's browser. However also this solution outsources some of the responsibility to the user: in order to have up-to-date information about the attendance status every nurse must ensure that the queue is empty when their shift ends.

5.4.6 Storing and Receiving Presence Markings on the Backend

Due to the simplicity of the offline support required for Päikky, most of the code related work was done on the Kindergarten UI only. Only offline supported feature which needed modifications to the backend was the ability for the user to create Presence Markings without active Internet connection at the time. This required some changes to be done on the mechanisms which receives and stores the Presence Markings coming from the Kindergarten UI.

Before the offline support Kindergarten UI did not send timestamps of the created Presence Markings to the backend. Timestamps were created on the backend at the

time when the requests from the Kindergarten UI were received.

To allow the postponed sending of the Presence Markings created in the past, the responsibility of creating the timestamps for the markings had to be moved from the backend to the Kindergarten UI. Since the client's clock can be set to anything, this created a need for a mechanism that could synchronize the clocks between the backend and the application.

Since the precision required by the application area is closer to a minute than a second, the client-backend clock synchronization could be done with a quite simple and straightforward approach. The same ping functionality which is used to determine the network status has also the ability to measure the time spent from sending the request from the client to the receiving of the backend's response. When this interval is known, the offset of the clocks on the application and on the backend can be determined. The backend returns the timestamp when the ping request is received on the backend to the application. When the measured duration of the request is split into half, the application can determine what is the offset between its and the backend's clocks. This offset is saved to the local storage, and added to each presence markings timestamp.

Due to the nature of TCP protocol (on which HTTP packets are delivered), the exact amount of spent on receiving the response from server can not be measured. Due to handshakes and other required precautions TCP does, more time is spent on the sending and delivering of the request than what it takes the response to arrive from the backend to the application. This creates the basic problem why the clock synchronizing can never be exact with this method. Päikky is configured to allow synchronizing result only from pings done within a three second timeframe, and as stated above, this is easily within the required precision for this use case. For applications where precise clock synchronization would be required, usage of a more sophisticated protocol engineered specially for that purpose would be required. Protocol of choice for that kind of use case could be the *Network Time Protocol (NTP)* [37].

5.4.7 Duplicating the Presence State Machine

Before the offline support the backend also validated the state transfer for the Kindergarten UI. If the new Presence type requested by the application was invalid, the backend would have returned a legal state, which the Kindergarten UI would have applied to the current person. From the user experience point of view this could be seen as a slight lag on the changing of the person's state when the new Presence marking was created. Kindergarten UI at time would not have known what the new state would be and therefore was forced to wait for the backend's response.

In order to enable the Kindergarten UI to know the allowed possible state transfers and to determine the new Presence state. This resulted as a need for duplicating the Presence State Machine from the backend to the Kindergarten UI. Duplicating it created some redundancy to the backend and Kindergarten UI codebases. Having the Presence state machine on both components of Päikky was inevitable, since as the keeper of the master data, removing the state validating from the backend would be shortsighted. Requests made to the API can be tampered with, and the actual payload has to be always validated by the backend. However having the same validation in two places creates a challenge for the developers: both have to be kept in sync in order for the system to operate properly.

On the positive side, the lag in the user interface with the Presence state transfer got minimized with the added functionality, since new state could be determined on the browser instantly without having to wait for the backend interaction's result.

6. EVALUATION

This chapter goes through the results got from the interviews. Solutions and compromises done on the technical side are also evaluated critically.

6.1 User Interviews

This section covers the analyzed results from the interviews. The main goal is to validate the prediction that the user experience of the offline mode is supporting the actual use cases on the kindergartens. Full transcriptions of the interviews are not available.

The interviews were conducted in two different kindergartens, both located in Pirkanmaa region of Finland. They are referred later only as *Kindergarten A* and *Kindergarten B*. Both kindergartens were fairly similar when looked by the main characteristics, both shares almost identical build years and have the ability to serve same amount of daycare children.

All of the users interviewed were full-time kindergarten nurses. Average age for the interviewees was 36,5 years. Youngest interviewee was 31 years old and the oldest was 52 years old. Both kindergartens where interviews were executed were opened fairly recently making the average length of employment only about 1,5 years (exact length of the employment periods were not questioned). Nevertheless there were lots of experience from the day care sector within the interviewees; the oldest interviewee had started their career in this field of operation in 1997.

The interviewees are not referred by their real name or the kindergarten they work in. Referring them is done by a randomly generated[1] number from a range from two to ten. The linking between interviewees or kindergartens and the generated numbers is not provided.

Interviewees were asked about what kind of phone they own, and how they would describe their skills on taking advantage from the phone. This aimed at charting the readiness and the base level of the user in using a digital service via smartphone. These basic characteristics are presented in Table 6.1, where each interviewee is indexed by the generated random number.

Only one of the interviewees did not have a phone that could not be described as a smartphone. They were also the only one who directly stated that their skills

[1] http://random.org

Table 6.1: Summary of the interviewees

#	Age	Own smartphone	Own description of smartphone usage skills
8	31 years	Samsung Galaxy S4	*"Basic user"*
2	31 years	Nokia Lumia 925	*"Advanced user"*
5	55 years	*"Some old Nokia"*	*"I can only call and send SMS's"*
8	33 years	*"Lumia Something"*	*"Fairly good"*

on using modern devices were fairly low. Other interviewees had smartphones, but only one had a smartphone sharing the same operating system used by Päikky. That correspondent was also the only one who described themselves as an advanced user of a smartphone. Other two stated that they can achieve basic functionality without much hesitation, but everything beyond that would be a challenge from some magnitude.

The mobile network coverage varied a lot between the kindergartens. Kindergarten A was said to have fairly good reception on most of the places, while having couple of dead spots. In the Kindergarten B, the reception was reported to be very bad or almost nonexistent, and this was also noticed by the interviewer when arriving in the premises.

6.1.1 General Feedback regarding Päikky Usage

In general the interviewees stated that they are satisfied with Päikky, and that it makes a lot of the daily activities easier. For example looking for children's parents' phone numbers from Päikky is a really straightforward and simple task in contrast to the old way of finding them from paper archives.

Nevertheless neither of the kindergartens did use Päikky as the only method of recording children attendance on the daycare. Both kindergartens created markings also in writing to the *care group diary*. Goings and leavings did not get marked to the diary as precisely as they get to Päikky though; no time markings were written into it. The diary logs could be seen more of an coexistence of a legacy system than a direct backup method. The diaries have existed long before the implementation of Päikky, since they were the method for tracking attendance prior it.

A non-surprising result from the interviews was that each of the interviewees stated the logging of children in and out as the activity they do the most with Päikky. This was said to be done tens of times per day.

At the start of the Päikky usage parents asked quite often instructions from the nurses. Interviewees stated that this stopped shortly thereafter. Mostly this was caused by the nurses' instructions which told the parents to contact the client organization's help desk in case of issues. In day-to-day interaction with parents the interviewees said that Päikky did not usually come up as an topic, unless there were

some issues with it. Usually the issue was parents who forgot to add care plans for their children.

Interviewees from Kindergarten A said that they have successfully moved all the communication between the homes and the kindergarten to be done via Päikky. Kindergarten B also used Päikky as their primary communication channel, but on addition to that they had standard procedure on communicating via paper which was used on some occasions.

All the interviewees told that within the nurses Päikky is a daily topic. Interviewees number 9, 2 and 5 stated that sometimes they vent their frustration with blaming Päikky out loud to fellow colleagues. More often talks relating to Päikky were regarded syncing with colleagues that did they correct some noticed inaccuracy on the attendance data or something else relating to correction of the markings.

Interviewee number 2 said that in their care group Kindergarten UI was often used on a desktop computer. Also the standard way of using it via smartphone was in use. The desktop computer was mostly in use due the easiness of the usage, and the central location of the said computer.

Interviewee number 8 stated that if they run against any issues with Päikky, they usually try to find someone else to solve them. Other interviewees could not remember issues with Päikky that they could not handle. The only exception to this was the case with the Kindergarten A's devices, that would seemingly load Päikky in an infinite loop. In situations like that the loader icon would not disappear after user started Kindergarten UI from the home menu, making the using of the application impossible. This was fixed on an update to the system later on.

6.1.2 User Experience of Offline Mode Implementation

All the interviewees stated that they would always notice if the application goes to offline mode. The offline mode header was said to be too eye-catching to be missed.

When the application entered offline mode, the common model of operation was just to wait for the connectivity to be reached again and keep on using the application on the offline mode as best as they can. Interviewees were asked if they had a habit of going to a specific place in order to achieve better reception, but this was said to be impossible since the nurses found themselves usually in a situation where they where the currently only supervisor for the children, and therefore they were unable to move. One interviewee said that they sometime try to move the device higher while trying to get a better reception.

Interviewees from the Kindergarten A reported that the Kindergarten UI can be on the offline mode for several minutes once it has entered it. Gaps on the Internet connectivity and the time for the application being on the offline mode were reported to be up to 15 minutes in length. Interviewees from the Kindergarten B – which had

better general coverage of mobile broadband network – said that the usual length for the offline mode being active was only couple of minutes. When asked directly about the speculated average length, both interviewees supposed that it would be near one minute.

In general the interviewees were satisfied to the offline mode regarding the key activity for Kindergarten UI, the logging in and out of kindergarten personnel. Interviewees had a consensus about the offline mode supporting their use case well; if the connection was nonexistent, they could still mark the children and the other nurses in or out, and the application would save them after the connectivity was reached again. If the offline mode was active at the end of the day, it was experienced as somehow burdensome. The client organization has instructed the nurses not to turn off the device when the offline mode is active, since not all the data is sent to the backend yet on that case. They have also been instructed to log out from the application and shut down the device when the kindergarten closes, so if the offline mode is active when they should end their workday and leave, they must wait for the Internet connectivity before they are allowed to leave. This was reported to happen only rarely, however.

Offline support also made the logging of kindergarten children and personnel more faster and easier, mostly due to the speed of the response got from clicking a button in the UI. This is a direct result of duplicating the state machine from the backend also to the Kindergarten UI.

Some inconvenience was experienced on how the offline mode affected the total head count of the kindergarten stated by the application. Since with the offline mode the devices can hold the data for long period of time without sending it to the backend, and therefore the information about changes in attendance would not get propagated into other devices. This resulted in an extra effort required especially on the morning and on the late afternoon, when most of the comings and leavings happens, and the information about the current headcount present is important. Usually the nurses kept the total headcount on their mind manually, because the amount implied by the application was felt too imprecise for real usage. Nevertheless it was said to be used as a general guide about the current amount of persons present.

Some randomly appearing issues on the application were reported by the inteviewees, but they were not identifiable as a direct errors on the functional logic of the offline mode. They seemed to be related on the startup procedure of application and how the user interface is initialized. Once found, these errors were left to a lower emphasis on the interview.

All of the interviewees had used Päikky prior to the existence of the offline mode. All of them stated that the offline mode has made their usage of Päikky more easier and less frustrating. The interviewees also said that even with its flaws, Päikky has

made their job easier, and they would not stop using it even if they were offered a chance on that.

6.1.3 User Experience of Limited Feature Set

As described in the previous chapters, once the application enters the offline mode, some of the features are disabled from the users. The interviewees did not find this distracting. The key activity being possible on both online and offline mode covered the major part of use cases described by the interviewees.

When asked about tasks that the interviewees had to have postponed due to Internet connection shortages and the limited feature set of the offline mode, they had hard time thinking of one. Two from the interviewees described writing a message to be one of the use cases that would some time need postponing due to the offline mode being active, but they did not think that this was frustrating. For some part this can be explained by the method they create messages with the application: in the Kindergarten B, where the reception was very bad, they use application on a desktop computer, which has physical access to the Internet.

6.1.4 Users' Understanding of the Offline Mode

Interviewees were asked a set of questions around technical theme on how do they understand the actual functionality behind the offline mode. These questions aimed at resolving if the concepts and abstractions done on the user interface were understandable by the users.

When asked to describe what causes the application going into offline mode, each of the interviewees successfully linked that into the Internet connection being lost. The understanding on what happens after that or why does it matter was not so well comprehended.

Every interviewee except one understood that the presence markings done while being on the offline mode were stored on the current device. The purpose of the size of the presence marking queue indicated on the user interface's header was also well comprehended. When asked about what happens when the application comes from the offline mode to the online mode, each of the interviewees correctly stated that the application sends the markings done to a place which none of them could not describe in more depth.

Each interviewee was asked what would happen on a imaginary scenario like this:

> "You are in the middle of morning rush, and lots of children are coming into the kindergarten. The application is on the offline mode, but you ignore that and keep on marking the children in. The rush settles, but the application is still on the offline mode. Then on a blink of an eye, a

> lightning strikes from the clear sky and turns your device into a pile of dust. What happened to the presence markings you created during the rush?"

None of the interviewees could give a straight, right answer to this scenario and about the data's destiny. With some aid, three of them came to the conclusion that the data was indeed lost. What was confusing was that even the interviewees understood that after the offline mode being active the presence markings created during it must be synchronized, they did not understand that the actual data was not located on the smartphones. The concept of the master data being on the backend was not clear to any of them, and this resulted in a kind of a hole on their logic. They recognized the importance of having an Internet connection, but they did not exactly know what is the main purpose for having the Internet connection.

6.2 Technical Effectiveness of the Implementation

With the scope being kept on mind, the implementation of the offline support can be considered to be successful.

Lot of the functionality implemented on the offline mode development would have been beneficial to the system and the users even if the actual offline support would not been on the Päikky's road map. For example effort towards the HTTP cache header optimization and the usage of application cache makes the Päikky's usage better for the user, resulting smaller data bandwidth usage and faster starting times.

The missing merge functionality on the backend is a downside, but that was a known compromise done in order to save available development resources during the creation of the offline support. The lack of this feature is covered with the instructions given to the users by the client organization. The nature of the general use case of the application also makes the nonexistence of this feature less meaningful, since most of the time care group's children are handled on a dedicated device. This makes the possibility of conflicted data happening relatively small.

The outsourcing of some responsibilities to the user – such as the possible merge conflict solving – can be seen as a kind of half-baked solution. It is also unfortunate when looking from both the user experience point of view and from the technical perspective. However this was a known decision in order to save resources and simplify development, evaluating it further is beneficial regarding this thesis.

In this area of operation the caching of the data on the browser is not problematic, but this might not be the same for every case. No encryption would be possible for that data, since both the key for decrypting and the actual data would need to be delivered and stored on the same place. This would make efficient encrypting impossible. For strictly confidential data the approach used in Päikky with the API

response caching would not therefore be useful.

Otherwise no compromises were made that would have left the technical aspect of any feature partial. This leaves the main emphasis on evaluating the success of the offline support to be decided on the user experience point of view.

7. LESSONS LEARNED

This chapter introduces lessons learnt by the developers while implementing the offline support to Päikky. The points presented in the *design guidelines* section are factors that are recommendations for developing similar functionalities into other single-page applications. The discussion also contains speculation on points detected during the development and deployment of the offline support, but which were too vague for basing any recommendations on.

7.1 Design Guidelines

This section introduces guidelines recommended by the development team for other developers facing similar scenarios. These guidelines are literally *lessons learned* by the development team and factors that they will take into consideration on the future projects.

7.1.1 Recognize the Essential Feature Set for Offline Support

When creating offline support to an application, optimal approach is naturally to support the whole user experience no matter if there is network connectivity or not. But the experiences gotten from the interviews of Päikky users indicate that splitting the features into primary and secondary categories and after that providing offline support only for the primary ones is a viable solution. This allows streamlining the development process and decreasing the amount of new code required a lot by having the possibility of simply disabling the secondary features when the Internet connection is not available.

It is important to identify the key activities of the use case at hand. They are different between systems operating in different kind of domains. These activities share the basic aspect of being non-postponable actions. In Päikky's case they are activities which include the time factor being a key part of the activity, like the marking of children and nurses to the application. If this is not done exactly when the arrive to or the depart from the kindergarten happens, extra work is required.

The kindergarten nurses did not experience the limited feature set of offline mode in the Kindergarten UI to be problematic. Postponing sending messages and editing the presence data later on was natural to them, since these were either way tasks that they would do on occasions when they have a moment of spare time from the

actual work.

By recognizing the essential features for offline support a great user experience can be achieved without refactoring the whole codebase to comply with stoppages on the Internet connection.

7.1.2 Prepare for Possible Offline Support

As web applications are used more often as the implementation method of "serious" use cases, the need for cases when offline support is required can be expected to grow. When developing new applications and making design decisions regarding architecture, it is reasonable to at least keep the possibility of needing to implement offline support later on in mind.

On the browser environment a great level of the preparedness can be achieved with simple solutions. For example in Päikky it was very advantageous that all the interaction towards the backend was implemented through a single module (on Kindergarten UI that module was Backbone's *sync function*). Overriding it to use the local storage cache on the offline mode could be done in only single location on the code, in contrast to what it would have been if AJAX requests would have been made individually by different modules.

The most important part of a single-page application where this can be taken into account is the section of the system where data synchronizing happens. It is essential that there **is** a centralized point where this happens. By this a drop-in creation of offline support is made much more feasible. This can be seen as a good practice on web applications overall, even when the possible offline support is left out of consideration.

7.1.3 Use Application Cache

Use *application cache* regardless you are implementing offline support or not. At the same time know the limitations and pitfalls of it.

User experience of the application gets better when using the application cache. Proper usage of it results in form of shorter initializing times and faster loading of content when switching between the views on the application. Amount of data transfer required can also be reduced.

One should not try to get advantage from the application cache without proper exploration of the technique. The nature of the application's resources (image assets, JavaScript files) must also be studied and the cache manifest has to be tailored to match the unique characteristics of each application. In order to create an efficient production roll-out processes creating a build step to the deploy process is also required, if one does not already exist for the application.

The use of HTTP cache headers has also to be synchronized with the usage of cache manifest. Otherwise the cache invalidation in end users' browsers can become a major challenge, and without knowledge of the mechanisms involved developers can easily find themselves on a situation where the application's cache manifest itself is cached.

With all this taken into consideration, a well considered and planned usage of the application cache is encouraged to any kind of single-page application.

7.2 Discussion

No direct guidelines regarding user experience design could be pinpointed from the development process researched on this thesis. However a need for re-thinking the concepts presented on the user interface of the Kindergarten UI regarding the offline support was recognized. The current version – especially the concept of the *job queue* – can not be easily adapted intuitively. Instead instructions or trainings are needed.

As described above, users' understanding on the concepts of offline mode seemed to be inadequate. They are aware that the application being on the offline mode is a result of problems on the Internet connection, but they do not fully understand why it matters. The *client-server model* is not apparent to them, nor is the fact (originating from the unawareness of that model) that the master data exists in a single, centralized point instead of their smartphones. Without acknowledging the fact that the smartphones are merely a terminals where from the data can be looked and modified from, shaping a thorough understanding on the need for offline mode is challenging.

Abstracting the Internet connection status completely away from the end user is a handful. This depends on the domain of operation and the requirements set for the application, but some level of awareness regarding connection status is usually required from the user. Nevertheless the ultimate goal for offline support would be the removal of the network status from the user consideration, and maybe notice them about unsynchronized data only in the cases like exiting the application while unsaved changes exist.

With that being kept in mind, it is worth speculating on cases like Päikky where the status of the Internet connection is all the time implied to the user that **should the users be taught the basics of how web services work?** Knowing that they are based on the *client-server model* and understanding that model would result in a better context for understanding the offline mode in general and give the motivation for working with the inconvenience it causes. The users would not need to know the technical specifics of the system's architecture, but teaching them only the fact that the data exists somewhere else than on their smartphones would probably make a difference on understanding the main reason for the offline mode.

General naming of the concepts regarding offline support must also be reconsidered in cases where the Internet connection status must be implied to the user. The words *online* and *offline* (exactly the same term is used on all localizations of Päikky's user interface) are obvious for software engineers, but for a kindergarten nurse the meaning of those words might not be so clear. Choosing of terminology coming not from foreign origin would make assimilation of the concepts easier.

Therefore, and probably also more generally, further research on this topic could focus more on the human aspects of the development and give the technical point of view less significance.

8. CONCLUSIONS

Requirement for offline support in single-page applications is not extraordinary today, and the need for it can be expected to grow in the future. Developers should keep that in mind when developing web applications and especially when designing software architecture of single-page applications.

Offline support can be implemented to an application with different levels of extent. The support can also be implemented to an already existing application, and in case the architecture is well designed, this can be done without extraordinary effort. Providing offline support may not require major changes to the server side of the system, but can be applied primarily to the code ran on the browser environment.

Knowing the specifics of the domain area helps in identifying the essential key activities within the application. Supporting only these activities during an Internet connection blackout is usually sufficient and will not sacrifice the user experience disturbingly much. This allows restricting of the scope of offline support, resulting in a fewer resources required on the development effort.

New features on the HTML standards – such as *application cache* and *local storage* – forms a great toolbox for implementing offline support for single-page applications.

When looking from the user experience point of view, the technical methods introduced and evaluated in this thesis can also be beneficial in the development of single-page application in cases where there is no need for enabling usage of the application during a total Internet connection blackout. Applying these methods properly will result in a faster initialization times and smaller bandwidth usage than when using traditional approaches.

In cases where the status of the Internet connection can not be fully abstracted from the user's comprehension (only partial offline support is implemented), the concepts shown in the user interface must be considered thoroughly. For an average user the *client-server model* on which Internet is built might not be distinct at all, and that can make motivating them to observe or care for the network status challenging. In scenarios like these, instructing the users about the context of the web applications in general might be the best alternative.

REFERENCES

[1] StatCounter. Internet usage global stats. Available at: `http://gs.statcounter.com/#all-comparison-ww-monthly-201311-201411/` Referenced 04.12.2014.

[2] A. Berson. *Client-server architecture*. J. Ranade series on computer communications. McGraw-Hill, 1992.

[3] M. Jazayeri. Some Trends in Web Application Development. In *Future of Software Engineering, 2007. FOSE '07*, pages 199–213, May 2007.

[4] Antero Taivalsaari. Mashware: The future of web applications. Technical report, Sun Microsystems, Inc., Mountain View, CA, USA, 2009.

[5] Thomas Duebendorfer and Stefan Frei. Why silent updates boost security. *TIK, ETH Zurich, Tech. Rep*, 302, 2009.

[6] Linda Dailey Paulson. Building rich web applications with ajax. *Computer*, 38(10):14–17, October 2005.

[7] Jesse James Garrett. Ajax: A new approach to web applications. Available at: `https://courses.cs.washington.edu/courses/cse490h/07sp/readings/ajax_adaptive_path.pdf`, February 2005.

[8] Mark Masse. *REST API Design Rulebook*. "O'Reilly Media, Inc.", October 2011.

[9] Marco Casario, Peter Elst, Charles Brown, Nathalie Wormser, and Cyril Hanquez. HTML5 local storage. In *HTML5 Solutions: Essential Techniques for HTML5 Developers*, pages 281–303. Apress, January 2011. chapter 11-2 / s. 285.

[10] Roy Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[11] Roger L. Costello. Building web services the rest way. *URL: http://www.xfront.com/REST-Web-Services. HTML. Ultima Consulta*, 11:2007, 2007.

[12] Peter H. Carstensen and Kjeld Schmidt. Computer supported cooperative work: New challenges to systems design. In *K. Itoh (Ed.), Handbook of Human Factors*, pages 619–636, 1999.

[13] Richard Bentley, Wolfgang Appelt, Uwe Busbach, Elke Hinrichs, Douglas Kerr, Klaas Sikkel, Jonathan Trevor, and Gerd Woetzel. Basic support for cooperative work on the world wide web. *International journal of human-computer studies*, 46(6):827–846, 1997.

[14] John M. Carroll, Dennis C. Neale, Philip L. Isenhour, Mary Beth Rosson, and D. Scott McCrickard. Notification and awareness: synchronizing task-oriented collaborative activity. *International Journal of Human-Computer Studies*, 58(5):605–632, 2003.

[15] Milad Saad and Mary Lou Maher. Shared understanding in computer-supported collaborative design. *Computer-Aided Design*, 28(3):183–192, March 1996.

[16] Jaana Näsänen, Antti Oulasvirta, and Asko Lehmuskallio. Mobile media in the social fabric of a kindergarten. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 2167–2176, New York, NY, USA, 2009. ACM.

[17] Feng Qian, Kee Shen Quah, Junxian Huang, Jeffrey Erman, Alexandre Gerber, Zhuoqing Mao, Subhabrata Sen, and Oliver Spatscheck. Web caching on smartphones: Ideal vs. reality. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, MobiSys '12, pages 127–140, New York, NY, USA, 2012. ACM.

[18] Internet Live Stats. Number of Internet Users (2015). Available at: `http://www.internetlivestats.com/internet-users` Referenced 19.4.2015.

[19] Cor-Paul Bezemer and Andy Zaidman. Multi-tenant SaaS Applications: Maintenance Dream or Nightmare? In *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*, IWPSE-EVOL '10, pages 88–92, New York, NY, USA, 2010. ACM.

[20] Alan Hevner and Samir Chatterjee. *Design Research in Information Systems: Theory and Practice*. Springer Science & Business Media, June 2010.

[21] Alan R. Hevner. A three cycle view of design science research. *Scandinavian journal of information systems*, 19(2):4, 2007.

[22] Kalle A Piirainen and Rafael A Gonzalez. Constructive synergy in design science research: A comparative analysis of design science research and the constructive research approach. *Liiketaloudellinen Aikakauskirja*, (3-4):206–234, 2014.

[23] Dr Jonathan Lazar, Dr Jinjuan Heidi Feng, and Dr Harry Hochheiser. *Research Methods in Human-Computer Interaction*. John Wiley & Sons, February 2010.

[24] Robert K. Yin. *Case Study Research: Design and Methods*. SAGE Publications, Inc, Los Angeles, fifth edition, May 2013.

[25] W3.org. Hypertext transfer protocol 1.1 – header field definitions. Available at: `http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.9` Referenced 4.11.2014.

[26] Andrea Wharry. HTML5 Offline Technologies. Available at: `http://andreawharry.com/assets/img/portfolio/WharryAndreaFinalPaper.pdf` Referenced 19.4.2015., 2014.

[27] Web Hypertext Application Technology Working Group. HTML specification: Offline web applications. Available at: `https://html.spec.whatwg.org/multipage/browsers.html#offline` Referenced 27.10.2014.

[28] Bruce Lawson and Remy Sharp. *Introducing HTML5*. New Riders, October 2011.

[29] Jake Archibald. Application cache is a douchebag. Available at: `http://alistapart.com/article/application-cache-is-a-douchebag/`, May 2012.

[30] Mozilla Corporation. Using the application cache. Available at: `http://developer.mozilla.org/en-US/docs/Web/HTML/Using_the_application_cache/` Referenced 24.11.2014.

[31] jQuery Foundation. .ajaxError(), jQuery API documentation. Available at: `http://api.jquery.com/ajaxerror/` Referenced 4.11.2014.

[32] Mozilla Corporation. Document object model: window.navigator.onLine. Available at: `https://developer.mozilla.org/fi/docs/DOM/window.navigator.onLine/` Referenced 25.11.2014.

[33] W3.org. Web storage - the localStorage attribute. Available at: `http://www.w3.org/TR/webstorage/#the-localstorage-attribute/` Referenced 11.12.2014.

[34] Philip A. Laplante. *Dictionary of Computer Science, Engineering and Technology*. CRC Press, December 2000.

[35] Daniel P. Friedman and David Stephen Wise. *The impact of applicative programming on multiprocessing.* Indiana University, Computer Science Department, 1976.

[36] jQuery Foundation. deferred.promise(),jQuery API documentation. Available at: `http://api.jquery.com/deferred.promise/` Referenced 20.11.2014.

[37] D.L. Mills. Internet time synchronization: the network time protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, October 1991.

A. APPENDIX: INTERVIEW TEMPLATE

Haastattelu Päikky-järjestelmän offline-tuesta

Vastauksia käytetään nimettömänä Miro Niemisen diplomityön aineistona. Vastauksien perusteella pyritään jatkokehittämään Päikky-järjestelmää entistä käyttäjäystävällisemmäksi.

Haastattelu nauhoitetaan ääninauhurilla litterointia varten, ja nauhoitukset tuhotaan diplomityön valmistuttua.

Jos jokin kysymys tuntuu arveluttavalta tai epämiellyttävältä, siihen voi jättää vastaamatta.

Mihinkään kysymykseen ei ole olemassa oikeita vastauksia, vaan haen mahdollisimman omakohtaisia kokemuksia asiasta.

1. Kerro lyhyesti työnkuvasi, ja kuinka kauan olet ollut töissä tässä päiväkodissa? Saanko käyttää tätä tietoa DI-työssäni?
2. Taustaa: ikä, käytössä oleva puhelin, teknologiaosaaminen
3. Mihin käytit viimeksi Päikkyä?
4. Kuvaile yleisimpiä tehtäviä, mitä teet Päikyllä päivittäin?
5. Mikä on viimeisin ongelma, mikä Sinun Päikky-käytössäsi on ilmennyt? Miten selvisit siitä?
6. Mitä olette keskustelleet Päikystä kolleegoiden kesken?
7. Onko Päikky ainoa kirjanpitoväline vai käytättelö jotain muuta sen lisäksi?
8. Kuinka ymmärrät sanan offline-moodi?
9. Jos Päikky menee offline-moodiin mitä mielestäsi silloin tapahtuu?
10. Oletko tietoinen Päikyn offline-moodista?
11. Näytä ja kerro kuinka Päikky mielestäsi kertoo onko se online vai offline-moodissa?
12. Huomaatko aina, jos Päikky menee offline-moodiin?

13. Kuinka usein käyttäessäsi Päikkyä järjestelmä menee offline-moodiin?
14. Mitä teet jos huomaat että Päikky menee offline-moodiin?
15. Onko jotain erityisiä paikkoja jossa Päikky menee offline-moodiin?
16. Kuinka pitkiä aikoja järjestelmä pysyy offline-moodissa kerran siihen mentyään?
17. Rajoittaako offline-moodi Sinun Päikyn käyttöä mitenkään? Joudutko lykkäämään jotain tehtäviä, joita normaalisti tekisit heti?
18. Mikä on viimeisin ongelma, mikä Sinulla on ollut Päikyn offline-moodiin liittyen?
19. Mitä olette keskustelleet kolleegoiden/vanhempien kanssa offline moodista?
20. Oletko käyttänyt Päikkyä ennen offline-moodin olemassaoloa?
21. Jos olet, niin miten offline-moodin olemassaolo on muuttanut Päikyn käyttöäsi?
22. Kerro omin sanoin, mitä käsität tapahtuvan Päikyn mennessä offline-moodiin
23. Jos merkitset Päikyssä lapsen saapuneeksi offline-moodin ollessa päällä, ja hetki tämän jälkeen salama iskee laitteeseesi muuttaen sen kasaksi tuhkaa, mitä oletat tapahtuvan juuri tekemällesi lapsen sisäänkirjaukselle?
24. Kerro omin sanoin, mitä käsität tapahtuvan Päikyn poistuessa offline-moodista
25. Haluatko jatkaa Päikyn käyttöä?
26. Kerro terveisesi ja kehitysehdotuksesi Päikky-järjestelmän kehittäjille

Kiitos!